Etape 1 : Exploration des données

Dans cette étape, nous explorons les données du dataset dataset_big_patent_v1.json suivant différents axes :

- analyses basiques sur le contenu des documents
- analyses des différentes sections composant le brevet

In [1]:
```
import json
import seaborn as sns
from nltk.corpus import stopwords
from collections import  Counter
import matplotlib.pyplot as plt
import numpy as np
```

Chargement du dataset

In [2]:
```
with open('/Users/mbp004/dev/test_yxir/data/dataset_big_patent_v1.json') as
    data = json.load(f)
```

Analyses exploratoires basiques

On regarde des indicateurs de base sur les donnees (brevet+abstract) :

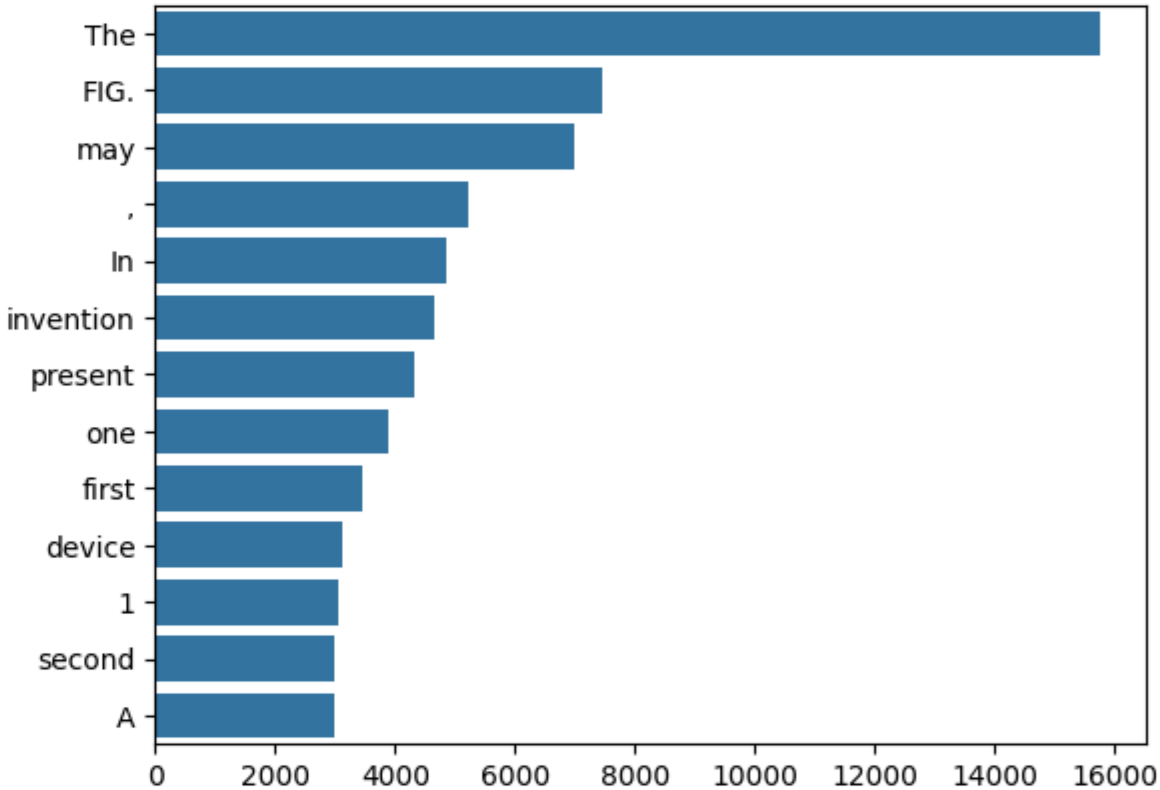
- mots les plus frequents
- longueur des documents

In [3]:
```
# Construction simplifiee d'un corpus pour l'analyse
corpus = []
for i in range(len(data)):
    corpus.append(data[i]['negative'])
    corpus.append(data[i]['pos'])
```

In [4]:
```
# Affichage des mots les plus frequents, exceptes les stop words
stop=set(stopwords.words('english'))
corpus_words = []
for text in corpus:
    new= text.split()
    corpus_words+=new
counter=Counter(corpus_words)
most=counter.most_common()
x, y=[], []
for word,count in most[:40]:
    if (word not in stop):
        x.append(word)
        y.append(count)
```

```
sns.barplot(x=y,y=x)
```

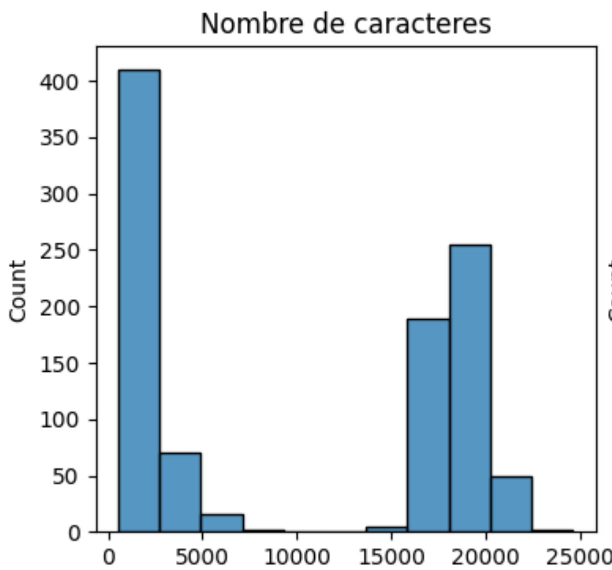
Out[4]: <Axes: >

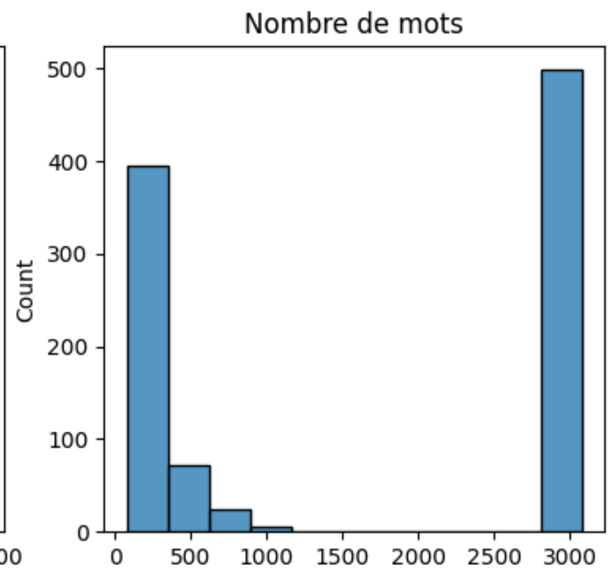


In [5]:

```
# Repartition sur la taille des documents
corpus_character_length = [len(text) for text in corpus]
corpus_words_length = [len(text.split()) for text in corpus]
fig, ax = plt.subplots(1,2)
fig.set_figheight(4)
fig.set_figwidth(9)
sns.histplot(data=corpus_character_length, ax=ax[0])
sns.histplot(data=corpus_words_length, ax=ax[1])
ax[0].set_title('Nombre de caracteres')
ax[1].set_title('Nombre de mots')
fig.show()
```

```
/var/folders/7d/q4n92n2n2_d49gm_83s92yyh0000gp/T/ipykernel_3112/3284534153.p
y:11: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be sh
own
  fig.show()
```





Focus sur les differentes sections des brevets

On infère les différentes sections contenues dans les brevets et on fait quelques stats sur les sections :

- listing des sections trouvées
- Sur les documents : Combien d'abstracts VS contenus de brevets
- longueur moyennes des différentes sections

In [6]:
```
# Inference des differents noms de section possible dans un brevet
sections_candidates = []

for text in corpus:
    list_candidate = []
    for word in text.split():
        if (word.isupper() and len(word)>1):
            list_candidate.append(word)
        else:
            if len(list_candidate)>0:
                sections_candidates.append(' '.join(list_candidate))
            list_candidate = []

sections_counter = Counter(sections_candidates)
# A partir de ce counter on infere les noms de sections suivants:

list_all_patent_sections = [
    'CROSS-REFERENCE TO RELATED APPLICATIONSDETAILED DESCRIPTION OF THE PREF
    'STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT',
    'BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS',
    'DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT',
    'STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH',
    'DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS',
    'DETAILED DESCRIPTION OF THE INVENTION FIG.',
    'CROSS REFERENCES TO RELATED APPLICATIONS',
    'BRIEF DESCRIPTION OF THE DRAWING FIGURES',
```

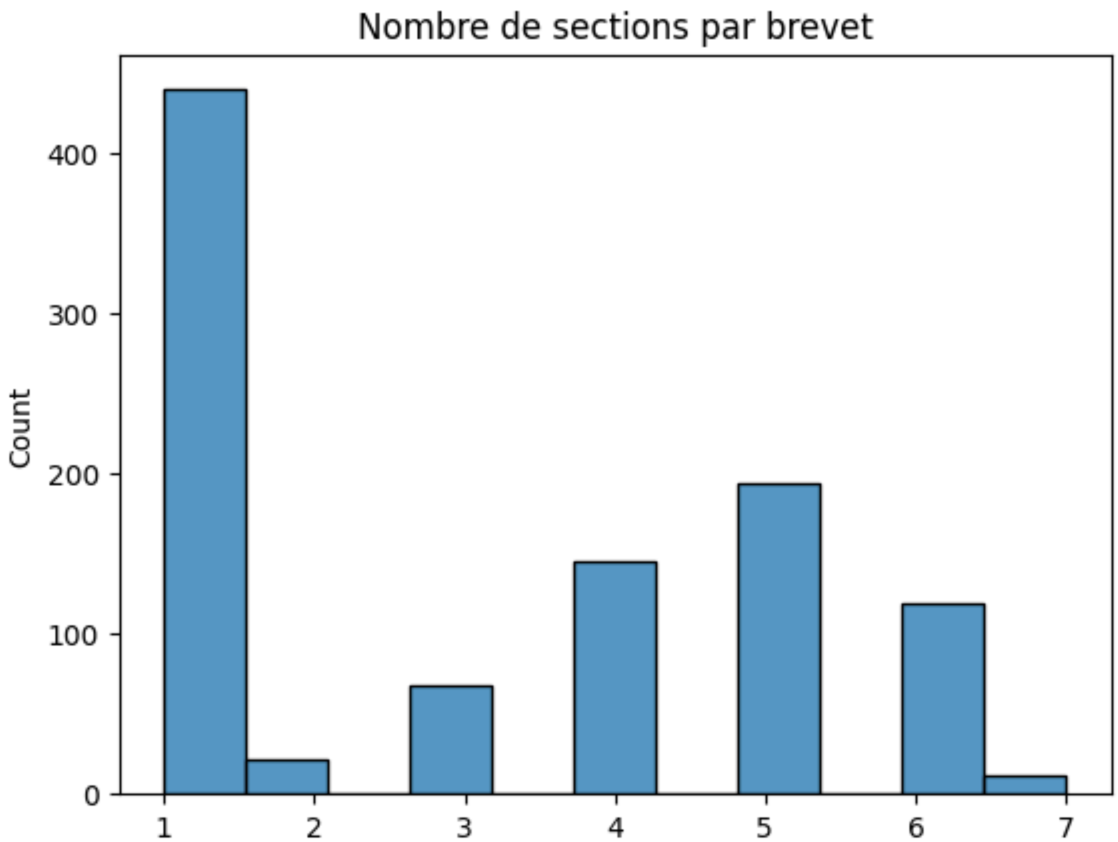
```python
    'CROSS REFERENCE TO RELATED APPLICATIONS',
    'DESCRIPTION OF THE PREFERRED EMBODIMENT',
    'DETAILED DESCRIPTION OF THE EMBODIMENTS',
    'BACKGROUND AND SUMMARY OF THE INVENTION',
    'CROSS-REFERENCE TO RELATED APPLICATION',
    'CROSS REFERENCE TO RELATED APPLICATION',
    'DETAILED DESCRIPTION OF THE INVENTION',
    'OBJECTS AND SUMMARY OF THE INVENTION',
    'DETAILED DESCRIPTION OF EMBODIMENTS',
    'BRIEF DESCRIPTION OF THE DRAWING(S)',
    'DESCRIPTION OF PREFERRED EMBODIMENT',
    'BRIEF DESCRIPTION OF THE INVENTION',
    'BRIEF DESCRIPTION OF THE DRAWINGS',
    'DESCRIPTION OF THE DRAWINGS FIG.',
    'BRIEF DESCRIPTION OF THE FIGURES',
    'TECHNICAL FIELD OF THE INVENTION',
    'BRIEF DESCRIPTION OF THE DRAWING',
    'BRIEF SUMMARY OF THE INVENTION',
    'DESCRIPTION OF THE RELATED ART',
    'BRIEF DESCRIPTION OF DRAWINGS',
    'DESCRIPTION OF THE PRIOR ART',
    'DESCRIPTION OF THE INVENTION',
    'BRIEF DESCRIPTION OF FIGURES',
    'BACKGROUND OF THE INVENTION',
    'DESCRIPTION OF THE DRAWINGS',
    'BACKGROUND TO THE INVENTION',
    'DISCLOSURE OF THE INVENTION',
    'DESCRIPTION OF RELATED ART',
    'DESCRIPTION OF THE FIGURES',
    'THE FIELD OF THE INVENTION',
    'SUMMARY OF THE DISCLOSURE',
    'SUMMARY OF THE INVENTION',
    'OBJECTS OF THE INVENTION',
    'BACKGROUND AND PRIOR ART',
    'DISCUSSION OF PRIOR ART',
    'BACKGROUND OF INVENTION',
    'FIELD OF THE DISCLOSURE',
    'FIELD OF THE INVENTION',
    'DETAILED DESCRIPTION',
    'RELATED APPLICATIONS',
    'SUMMARY OF INVENTION',
    'RELATED APPLICATION',
    'FIELD OF INVENTION',
    'BRIEF DESCRIPTION',
    'TECHNICAL FIELD',
    'TECHINCAL FIELD',
    'BACKGROUND ART',
    'DESCRIPTION',
    'BACKGROUND',
    'SUMMARY'
]

# on sauvegarde pour la suite
json_dict = {'section_names': list_all_patent_sections}

with open("../data/patent_sections.json", "w") as outfile:
```

```
    json.dump(json_dict, outfile, indent = 6)
    outfile.close()
```

In [7]:
```
# Comptage du nombre de section par brevet

def transform_patent_text_to_list_sections(text):
    def split_by_section(list_sections, text):
        if len(list_sections) == 0:
            return [text]
        else:
            section = list_sections[0]
            text_split = text.split(section)
            if len(text_split) == 1:
                return split_by_section(list_sections[1:], text)
            else:
                list_return = split_by_section(list_sections[1:], text_split
                for sub_text in text_split[1:]:
                    list_return.append({'section':section})
                    list_return+=split_by_section(list_sections[1:], sub_tex
                return list_return

    splitted_sections = split_by_section(list_all_patent_sections, text)
    splitted_sections = [el for el in splitted_sections if el!='']

    patent_sections = []
    candidate_section = None
    for el in splitted_sections:
        if candidate_section is None:
            if isinstance(el, str):
                candidate_section = {'section': '', 'content':el}
                patent_sections.append(candidate_section)
                candidate_section = None
            else:
                candidate_section = el
        else:
            if isinstance(el, str):
                candidate_section['content'] = el
                patent_sections.append(candidate_section)
                candidate_section = None
            else:
                patent_sections.append(candidate_section)
                candidate_section = el
    return patent_sections

corpus_patent_sections = {}
for i in range(len(corpus)):
    text = corpus[i]
    patent_sections = transform_patent_text_to_list_sections(text)
    corpus_patent_sections[i] = patent_sections
```

In [8]:
```
list_nb_sections = [len(corpus_patent_sections[i]) for i in range(len(corpus
ax = sns.histplot(data=list_nb_sections)
ax.set_title('Nombre de sections par brevet')
```

Out[8]: Text(0.5, 1.0, 'Nombre de sections par brevet')




```
In [16]: # Nombre de contenu de brevet VS abstract
nb_abstract = 0
for i in corpus_patent_sections.keys():
    patent_sections = corpus_patent_sections[i]
    if (len(patent_sections) == 1) and (patent_sections[0]['section'] == ''):
        nb_abstract+=1
nb_patent_content = len(corpus_patent_sections) - nb_abstract
print('Au total dans le jeud de donnees:')
print('    Nombre de contenus de brevet : {}'.format(nb_patent_content))
print('    Nombre d\'abstracts de brevet: {}'.format(nb_abstract))
```

```
Au total dans le jeud de donnees:
    Nombre de contenus de brevet : 564
    Nombre d'abstracts de brevet: 434
```

```
In [146…: Counter(list_nb_sections)
```

```
Out[146…: Counter({1: 440, 5: 194, 4: 145, 6: 119, 3: 68, 2: 21, 7: 11})
```

```
In [166…: # Stats sur les sections les plus longues
dic_len_sections = {key:[] for key in list_all_patent_sections+['']}

for i in range(len(corpus_patent_sections)):
    list_sections = corpus_patent_sections[i]
    for section in list_sections:
        key = section['section']
        value = len(section['content'].split())
        dic_len_sections[key].append(value)
```

```
for key in dic_len_sections:
    if len(dic_len_sections[key]) == 0:
        dic_len_sections[key] = 0
    else:
        dic_len_sections[key] = np.mean(dic_len_sections[key])

list_len_sections = [(key, value) for key,value in dic_len_sections.items()]

sorted(list_len_sections, key = lambda x : -x[1])
```

```
Out[166… [('DISCLOSURE OF THE INVENTION', 2188.5),
 ('OBJECTS AND SUMMARY OF THE INVENTION', 1688.5),
 ('DETAILED DESCRIPTION OF EMBODIMENTS', 1567.0),
 ('DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT', 1502.1797752808989),
 ('DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS', 1470.0),
 ('DETAILED DESCRIPTION OF THE INVENTION', 1466.7364864864865),
 ('DETAILED DESCRIPTION', 1449.516393442623),
 ('DESCRIPTION OF PREFERRED EMBODIMENT', 1405.6666666666667),
 ('DESCRIPTION OF THE PREFERRED EMBODIMENT', 1374.7115384615386),
 ('BACKGROUND AND SUMMARY OF THE INVENTION', 1204.4),
 ('DETAILED DESCRIPTION OF THE INVENTION FIG.', 1084.3333333333333),
 ('DESCRIPTION OF THE FIGURES', 943.5714285714286),
 ('BACKGROUND TO THE INVENTION', 882.8),
 ('DESCRIPTION OF THE INVENTION', 813.4285714285714),
 ('BACKGROUND AND PRIOR ART', 794.0),
 ('BRIEF SUMMARY OF THE INVENTION', 755.375),
 ('DISCUSSION OF PRIOR ART', 744.2857142857143),
 ('BACKGROUND OF INVENTION', 739.8),
 ('DESCRIPTION OF THE PRIOR ART', 734.9),
 ('DETAILED DESCRIPTION OF THE EMBODIMENTS', 734.6666666666666),
 ('DESCRIPTION OF RELATED ART', 720.6666666666666),
 ('SUMMARY OF THE INVENTION', 713.3575418994413),
 ('SUMMARY OF INVENTION', 664.125),
 ('SUMMARY', 649.9425287356322),
 ('BACKGROUND OF THE INVENTION', 645.1748633879781),
 ('DESCRIPTION', 605.3636363636364),
 ('DESCRIPTION OF THE RELATED ART', 581.0),
 ('BACKGROUND', 519.7328244274809),
 ('BACKGROUND ART', 512.0),
 ('BRIEF DESCRIPTION OF DRAWINGS', 502.2857142857143),
 ('SUMMARY OF THE DISCLOSURE', 497.44444444444446),
 ('BRIEF DESCRIPTION OF THE INVENTION', 350.0),
 ('BRIEF DESCRIPTION OF THE DRAWING(S)', 301.0),
 ('DESCRIPTION OF THE DRAWINGS', 281.4782608695652),
 ('BRIEF DESCRIPTION', 278.14285714285717),
 ('BRIEF DESCRIPTION OF THE FIGURES', 263.6470588235294),
 ('BRIEF DESCRIPTION OF THE DRAWINGS', 254.85521885521885),
 ('BRIEF DESCRIPTION OF FIGURES', 240.0),
 ('BRIEF DESCRIPTION OF THE DRAWING FIGURES', 214.4),
 ('BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS',
  204.84615384615384),
 ('', 175.13058419243987),
 ('BRIEF DESCRIPTION OF THE DRAWING', 153.4),
 ('DESCRIPTION OF THE DRAWINGS FIG.', 145.0),
 ('OBJECTS OF THE INVENTION', 134.22222222222223),
 ('FIELD OF THE DISCLOSURE', 84.66666666666667),
 ('TECHNICAL FIELD OF THE INVENTION', 81.7),
 ('CROSS REFERENCES TO RELATED APPLICATIONS', 79.25),
 ('RELATED APPLICATIONS', 64.69565217391305),
 ('FIELD OF THE INVENTION', 62.883720930232556),
 ('FIELD OF INVENTION', 58.57692307692308),
 ('CROSS-REFERENCE TO RELATED APPLICATION', 52.536842105263155),
 ('CROSS REFERENCE TO RELATED APPLICATIONS', 48.205882352941174),
 ('CROSS REFERENCE TO RELATED APPLICATION', 45.888888888888886),
 ('TECHNICAL FIELD', 42.411764705882355),
 ('RELATED APPLICATION', 42.142857142857146),
```

```
 ('TECHINCAL FIELD', 27.0),
 ('THE FIELD OF THE INVENTION', 22.0),
 ('STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH', 12.6),
 ('STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT',
  5.153846153846154),
 ('CROSS-REFERENCE TO RELATED APPLICATIONSDETAILED DESCRIPTION OF THE PREFE
RRED EMBODIMENTS',
  0)]
```

In [18]:

```
# Comparaison de la longueur moyenne des contenu de brevet VS la longueur mo
longueur_abstracts = []
longueur_patents = []

for i in corpus_patent_sections.keys():
    patent_sections = corpus_patent_sections[i]
    if (len(patent_sections) == 1) and (patent_sections[0]['section'] == '')
        nb_words = len(patent_sections[0]['content'].split())
        longueur_abstracts.append(nb_words)
    else:
        nb_words = 0
        for j in range(len(patent_sections)):
            nb_words += len(patent_sections[j]['content'].split())
        longueur_patents.append(nb_words)

longueur_abstracts_mean = int(np.mean(longueur_abstracts))
longueur_patents_mean = int(np.mean(longueur_patents))

print('Longueur moyenne des abstracts : {} mots'.format(longueur_abstracts_m
print('Longueur moyenne des contenus de brevet : {} mots'.format(longueur_pa
```

```
Longueur moyenne des abstracts : 216 mots
Longueur moyenne des contenus de brevet : 2711 mots
```

Les enseignements qu'on en tire de cette analyse par section des brevets :

- Le jeu de donnees est presque equilibre entre les textes sous forme de contenu de brevet et les textes représentant les abstracts
- **Les abstracts sont en moyenne 12 fois moins longs que les contenus de brevet** : points de vigilance quand on fera des embeddings

Construction d'un dataset par section

Construction d'un dataset par section, qui sera utile pour la modelisation qui suivra. Le dataset aura cette structure :

{ 1: {'query': '...', 'pos': patent_sections_pos, 'negative': } ... }

In [28]:

```
dataset_section = {}
for i in range(len(data)):
    dataset_i = {}
    dataset_i['query'] = data[i]['query']
    dataset_i['pos'] = transform_patent_text_to_list_sections(data[i]['pos']
```

```
    dataset_i['negative'] = transform_patent_text_to_list_sections(data[i]['
    dataset_section[i] = dataset_i
```

In [31]:

```
with open("../data/dataset_patent_sections.json", "w") as outfile:
    json.dump(dataset_section, outfile)
    outfile.close()
```

Etape 2 : Modèle de base

- Construction d'une baseline Zero-Shot a partir d'un modèle LLM
- Finetuning de ce modèle à partir du dataset (contrastive loss)

In [1]:

```
import json
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import pickle

from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim
from sentence_transformers.quantization import quantize_embeddings
from sentence_transformers import losses
from sentence_transformers.readers import InputExample
from torch.utils.data import DataLoader
```

```
/Users/mbp004/dev/test_yxir/yxir/lib/python3.11/site-packages/threadpoolctl.
py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
```

Methode Zero-Shot

On prend un modele du MTEB leaderboard pour realiser les embeddings de documents On teste le modele suivant : mxbai-embed-large-v1, proposé par mixedbread ai. Le modèle a été entrainé sur la tache de STS (Semantic Textual Similarity)

In [2]:

```
# 1. Specify preffered dimensions
dimensions = 512

# 2. load model
model_name = 'intfloat/e5-small-v2'
# model_name = "mixedbread-ai/mxbai-embed-large-v1"
model = SentenceTransformer(model_name, truncate_dim=dimensions)
```

```
/Users/mbp004/dev/test_yxir/yxir/lib/python3.11/site-packages/huggingface_hu
b/file_download.py:1132: FutureWarning: `resume_download` is deprecated and
will be removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
```

In [3]:

```python
# Code utilisé sur Hugging Face
# For retrieval you need to pass this prompt.
query = 'Represent this sentence for searching relevant passages: A man is e

docs = [
    query,
    "A man is eating food.",
    "A man is eating pasta.",
    "The girl is carrying a baby.",
    "A man is riding a horse.",
]

# 2. Encode
embeddings = model.encode(docs)

# Optional: Quantize the embeddings
binary_embeddings = quantize_embeddings(embeddings, precision="ubinary")

similarities = cos_sim(embeddings[0], embeddings[1:])
print('similarities:', similarities)
```

```
similarities: tensor([[0.8732, 0.8448, 0.6409, 0.7452]])
```

Experience sur le dataset big_patent

On calcule les embeddings brutalement sur les contenus des brevets, abstracts et sur les queries. Le but est de voir ici si la similarité est plus grande entre la query et l'exemple positif qu'entre la query et le sample négatif

In [4]:

```python
with open('/Users/mbp004/dev/test_yxir/data/dataset_big_patent_v1.json') as
    data = json.load(f)
```

In [5]:

```python
list_all_embeddings = []
for i in tqdm(range(len(data)), desc ="Calcul des embeddings initiaux"):
    query = data[i]['query']
    pos_text = data[i]['pos']
    negative_text = data[i]['negative']

    docs = [
        query,
        pos_text,
        negative_text
    ]

    embeddings = model.encode(docs)
    list_all_embeddings.append(embeddings)
```

```
Calcul des embeddings initiaux: 100%|██████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [06:23<00:00,  1.30it/s]
```

In [13]:

```python
dic_all_embeddings = {'embeddings': list_all_embeddings}
with open('../data/list_all_embeddings_e5-small-v2.pickle', 'wb') as fh:
```

```
    pickle.dump(dic_all_embeddings, fh)
    fh.close()
```

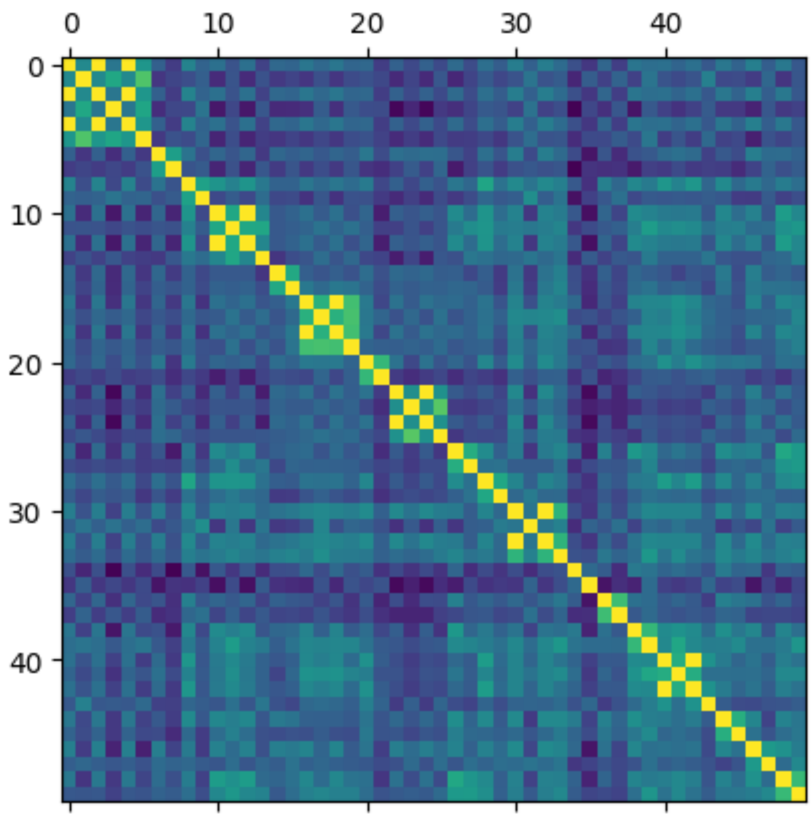
In [7]:
```
with open('../data/list_all_embeddings_e5-small-v2.pickle', 'rb') as fh:
    dic_all_embeddings = pickle.load(fh)
    fh.close()
list_all_embeddings = dic_all_embeddings['embeddings']
```

In [8]:
```
nb_good_embeddings = 0
for i in range(len(list_all_embeddings)):
    embeddings = list_all_embeddings[i]
    similarities = cos_sim(embeddings[0], embeddings[1:])
    sim_pos, sim_neg = similarities.flatten()
    if sim_pos > sim_neg :
        nb_good_embeddings+=1
perc_good_embeddings = round(100*nb_good_embeddings/len(list_all_embeddings)
print('Embeddings de documents compatibles avec la query: {}, {} %'.format(r
                                                                           p
```

```
Embeddings de documents compatibles avec la query: 370, 74.15 %
```

In [9]:
```
# Heatmap sur les similarites entre contenus de brevet
list_embeddings_heatmap = []
for i in range(len(list_all_embeddings)):
    list_embeddings_heatmap.append(list_all_embeddings[i][1])
    list_embeddings_heatmap.append(list_all_embeddings[i][2])

list_similarities_heatmap = []
for i in tqdm(range(len(list_embeddings_heatmap)), desc ="Calcul des similar
    list_similarities_heatmap_i = []
    for j in range(len(list_embeddings_heatmap)):
        sim = float(cos_sim(list_embeddings_heatmap[i], list_embeddings_heat
        list_similarities_heatmap_i.append(sim)
    list_similarities_heatmap.append(list_similarities_heatmap_i)
similarity_matrix = np.array(list_similarities_heatmap)

plt.matshow(similarity_matrix[0:50,0:50])
plt.show()
```

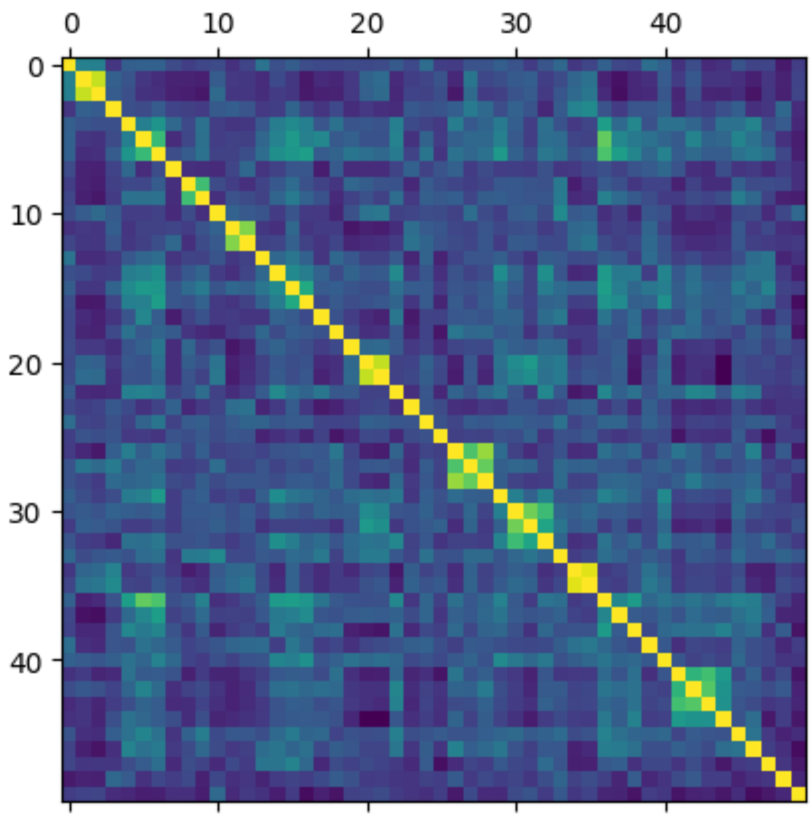
```
Calcul des similarites: 100%|███████████████████████████████████████████████
█████████████████████████████████████████████████████████████████████████████
████████████████| 998/998 [01:07<00:00, 14.87it/s]
```



Visuellement sur les embeddings de documents, on voit des carrés autour de la diagonale, ce qui traduit la construction du dataset où des questions successives autour du meme theme sont posees

In [10]:
```
# Heatmap sur les similarites entre queries
list_all_embeddings_queries = []
for i in range(len(list_all_embeddings)):
    list_all_embeddings_queries.append(list_all_embeddings[i][0])

list_similarities_queries_heatmap = []
for i in tqdm(range(len(list_all_embeddings_queries)), desc ="Calcul des sim
    list_similarities_heatmap_i = []
    for j in range(len(list_all_embeddings_queries)):
        sim = float(cos_sim(list_all_embeddings_queries[i], list_all_embeddi
        list_similarities_heatmap_i.append(sim)
    list_similarities_queries_heatmap.append(list_similarities_heatmap_i)
similarity_matrix_queries = np.array(list_similarities_queries_heatmap)

plt.matshow(similarity_matrix_queries[0:50,0:50])
plt.show()
```

```
Calcul des similarites sur les queries: 100%|███████████████████████████████
█████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:16<00:00, 30.54it/s]
```



On voit un peu cet effet sur les embeddings des queries meme si c'est moins flagrant. A priori ceci est du à :

- la taille des queries qui est tres faible donc plus facile a disciminer

Les pistes que ça nous amène à regarder :

- restreindre les documents a certaines sections, pour eviter que le vecteur d'embedding soit noyé dans pleins d'informations
- finetuner le modele avec les données de brevet

Performances en retrieval

Calcul des performances en retrieval sur une liste d'embeddings de type : [[emb_query, emb_pos, emb_neg]...] On cherche a calculer :

- le top_K_accuracy sur les labels positifs (a maximiser)
- le top_K_accuracy sur les labels negatifs (a minimiser)

In [35]:
```
def compute_top_K_accuracy_score(list_embeddings, K=5):
    '''
    Fonction pour calculer le top_K_accuracy score a partir d'une liste d'em
    [[emb_query, emb_pos, emb_neg]...]
    list_embeddings -- list, list des embeddings des query, positive, negati
    K -- int, le top K accuracy
    '''
    list_embeddings_query = [list_embeddings[i][0] for i in range(len(list_a
```

```
    list_embeddings_pos = [list_embeddings[i][1] for i in range(len(list_all
    list_embeddings_neg = [list_embeddings[i][2] for i in range(len(list_all

    nb_pos = 0
    nb_neg = 0
    for idx in tqdm(range(len(list_embeddings_query)), desc ="Calcul du top_
        query = list_embeddings_query[idx]

        similarities_pos = cos_sim(query, list_embeddings_pos).flatten().tol
        similarities_pos = [('pos_{}'.format(i), similarities_pos[i]) for i

        similarities_neg = cos_sim(query, list_embeddings_neg).flatten().tol
        similarities_neg = [('neg_{}'.format(i), similarities_neg[i]) for i

        similarities = similarities_pos+similarities_neg
        similarities = sorted(similarities, key = lambda x: -x[1])
        top_K_ids = [similarities[i][0] for i in range(K)]

        if 'pos_{}'.format(idx) in top_K_ids:
            nb_pos+=1
        if 'neg_{}'.format(idx) in top_K_ids:
            nb_neg+=1
    acc_K_pos = nb_pos/len(list_embeddings)
    acc_K_neg = nb_neg/len(list_embeddings)
    return acc_K_pos, acc_K_neg

acc_K_pos, acc_K_neg = compute_top_K_accuracy_score(list_all_embeddings, K=5
print('acc_K_pos : {} (a maximiser)'.format(acc_K_pos))
print('acc_K_neg : {} (a minimiser)'.format(acc_K_neg))
```

```
Calcul du top_K_accuracy score: 100%|██████████████████████████
███████████████████████████████████████████████████
███████████| 499/499 [00:25<00:00, 19.33it/s]
acc_K_pos : 0.9338677354709419 (a maximiser)
acc_K_neg : 0.7134268537074149 (a minimiser)
```

Finetuning du modele

On va tenter de finetuner le modele mxbai-embed-large-v1 sur les exemples de brevets

In [36]:
```
# Construction du dataset pour le finetuning
train_examples = []
for i in range(len(data)):
    anchor = data[i]['query']
    pos_example = data[i]['pos']
    neg_example = data[i]['negative']
    train_examples.append(InputExample(texts=[anchor, pos_example], label=1)
    train_examples.append(InputExample(texts=[anchor, neg_example], label=0)

train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=2)
```

In [37]:
```
# Training
train_loss = losses.ContrastiveLoss(model=model)
```

```
model.fit(
    [(train_dataloader, train_loss)],
    epochs=10,
)
```

```
Epoch:   0%|          | 0/10 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
Iteration:   0%|          | 0/499 [00:00<?, ?it/s]
```

In [38]:
```
# Recalculer les embeddings
list_all_embeddings_finetuning = []
for i in tqdm(range(len(data)), desc ="Calcul des embeddings apres finetunir
    query = data[i]['query']
    pos_text = data[i]['pos']
    negative_text = data[i]['negative']

    docs = [
        query,
        pos_text,
        negative_text
    ]

    embeddings = model.encode(docs)
    list_all_embeddings_finetuning.append(embeddings)

dic_all_embeddings = {'embeddings': list_all_embeddings_finetuning}
with open('../data/list_all_embeddings_e5-small-v2_finetuning.pickle', 'wb')
    pickle.dump(dic_all_embeddings, fh)
    fh.close()
```

```
Calcul des embeddings apres finetuning: 100%|██████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [06:20<00:00,  1.31it/s]
```

In [39]:
```
# Bonne classification
nb_good_embeddings = 0
for i in range(len(list_all_embeddings_finetuning)):
    embeddings = list_all_embeddings_finetuning[i]
    similarities = cos_sim(embeddings[0], embeddings[1:])
    sim_pos, sim_neg = similarities.flatten()
    if sim_pos > sim_neg :
        nb_good_embeddings+=1
perc_good_embeddings = round(100*nb_good_embeddings/len(list_all_embeddings_
print('Embeddings de documents compatibles avec la query: {}, {} %'.format(r
                                                                              p
```

```
Embeddings de documents compatibles avec la query: 499, 100.0 %
```

In [40]:
```python
# top_K_accuracy
acc_K_pos, acc_K_neg = compute_top_K_accuracy_score(list_all_embeddings_fine
print('acc_K_pos : {} (a maximiser)'.format(acc_K_pos))
print('acc_K_neg : {} (a minimiser)'.format(acc_K_neg))
```

```
Calcul du top_K_accuracy score: 100%|███████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:24<00:00, 20.68it/s]
acc_K_pos : 0.19438877755511022 (a maximiser)
acc_K_neg : 0.0 (a minimiser)
```

In [ ]:
```python

```

In [ ]:
```python

```

Etape 3 : Construction d'un modèle amélioré exploitant les sections

Pluôt que de prendre brutalement les embeddings des textes comme dans l'étape 2, nous faisons le travail suivant sur le text:

- pour chaque section, on chunk des phrases de tailles < input du LLM
- on réalise les embeddings de section par mean pooling des embeddings des phrases de section
- on réalise l'embedding du brevet par mean pooling des embeddings des sections

In [1]:
```
import json
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pickle

from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim
from sentence_transformers.quantization import quantize_embeddings
from sentence_transformers import losses
from sentence_transformers.readers import InputExample
from torch.utils.data import DataLoader
from transformers import AutoTokenizer
```

Creation des embeddings par groupe de phrases sur le brevet entier

On divise chaque section du brevet en phrases de nb_tokens < 512 pour réaliser les embeddings des phrases. Puis on moyenne les embeddings

In [2]:
```
with open('../data/dataset_patent_sections.json', 'r') as outfile:
    dataset_patent_section = json.load(outfile)
    outfile.close()
```

In [3]:
```
def get_text_sentences_from_tokenizer(text, tokenizer, dimensions = 512):
    '''
    Recuperation de la liste de phrases qui ont une taille inferieure a la c
    section -- str, le texte dont on veut recuperer la liste de phrases
    '''
    list_tokens = tokenizer.tokenize(text)
    list_sentences = []
    for i in range(int(len(list_tokens)/dimensions)+1):
        sentence = ' '.join(list_tokens[dimensions*i:dimensions*(i+1)])
        list_sentences.append(sentence)
```

```
    return list_sentences

def get_patent_text_sentences_from_tokenizer(patent, tokenizer, dimensions =
    '''
    Recuperation de la list de phrases constituant le brevet. Ici le brevt e
    patent -- list, representation du dictionnaire en sections
    '''
    list_sentences = []
    for i in range(len(patent)):
        text = patent[i]['content']
        list_sentences_text = get_text_sentences_from_tokenizer(text,
                                                                tokenizer,
                                                                dimensions =
        list_sentences += list_sentences_text
    return list_sentences
```

In [82]:

```
# Calcul des embeddings de tous les brevets comme moyenne des embeddings de

# model_name = "mixedbread-ai/mxbai-embed-large-v1"
model_name = 'intfloat/e5-small-v2'
tokenizer = AutoTokenizer.from_pretrained(model_name)
dimensions = 512
model = SentenceTransformer(model_name, truncate_dim=dimensions, revision=No

dataset_patent_section_embeddings = {}
for i in tqdm(range(len(dataset_patent_section)), desc ="Calcul des embeddin
    dataset_patent_section_embeddings[str(i)] = {}
    for key in ['pos', 'negative']:
        list_sentences = get_patent_text_sentences_from_tokenizer(dataset_pa
                                                                  tokenizer,
                                                                  dimensions
        embeddings = model.encode(list_sentences)
        patent_embedding = np.mean(embeddings, axis=0)
        dataset_patent_section_embeddings[str(i)][key] = patent_embedding
```

```
tokenizer_config.json:   0%|          | 0.00/314 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/711k [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/125 [00:00<?, ?B/s]
modules.json:   0%|          | 0.00/387 [00:00<?, ?B/s]
README.md:   0%|          | 0.00/67.8k [00:00<?, ?B/s]
sentence_bert_config.json:   0%|          | 0.00/57.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/615 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/133M [00:00<?, ?B/s]
1_Pooling/config.json:   0%|          | 0.00/200 [00:00<?, ?B/s]
```

```
Calcul des embeddings initiaux:   0%|
| 0/499 [00:00<?, ?it/s]Token indices sequence length is longer than the spe
cified maximum sequence length for this model (587 > 512). Running this sequ
ence through the model will result in indexing errors
Calcul des embeddings initiaux: 100%|███████████████████████████████
█████████████████████████████████████████████████████████████
█████████████| 499/499 [34:36<00:00,  4.16s/it]
```

In [87]:

```
# Ajout de l'embedding des query
for i in tqdm(range(len(dataset_patent_section)), desc ="Calcul des embeddin
    list_sentences = [dataset_patent_section[str(i)]['query']]
```

```
    embeddings = model.encode(list_sentences)
    patent_embedding = np.mean(embeddings, axis=0)
    dataset_patent_section_embeddings[str(i)]['query'] = patent_embedding
```

```
Calcul des embeddings des query: 100%|████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:10<00:00, 46.99it/s]
```

In [88]:
```
with open('../data/patent_sections_embeddings_e5-small-v2.pickle', 'wb') as
    pickle.dump(dataset_patent_section_embeddings, fh)
    fh.close()
```

In [4]:
```
with open('../data/patent_sections_embeddings_e5-small-v2.pickle', 'rb') as
    dataset_patent_section_embeddings = pickle.load(fh)
    fh.close()
```

In [5]:
```
# Performances -- classification
nb_good_embeddings = 0
for i in range(len(dataset_patent_section_embeddings)):
    embeddings = [
        dataset_patent_section_embeddings[str(i)]['query'],
        dataset_patent_section_embeddings[str(i)]['pos'],
        dataset_patent_section_embeddings[str(i)]['negative']
    ]
    similarities = cos_sim(embeddings[0], embeddings[1:])
    sim_pos, sim_neg = similarities.flatten()
    if sim_pos > sim_neg :
        nb_good_embeddings+=1
perc_good_embeddings = round(100*nb_good_embeddings/len(dataset_patent_secti
print('Embeddings de documents compatibles avec la query: {}, {} %'.format(r
                                                                          p
```

```
Embeddings de documents compatibles avec la query: 450, 90.18 %
```

In [10]:
```
# Performances -- top_K_accuracy
list_all_embeddings = []
for i in range(len(dataset_patent_section_embeddings)):
    emb_query = dataset_patent_section_embeddings[str(i)]['query']
    emb_pos = dataset_patent_section_embeddings[str(i)]['pos']
    emb_neg = dataset_patent_section_embeddings[str(i)]['negative']
    list_all_embeddings.append([emb_query, emb_pos, emb_neg])

def compute_top_K_accuracy_score(list_embeddings, K=5):
    '''
    Fonction pour calculer le top_K_accuracy score a partir d'une liste d'em
    [[emb_query, emb_pos, emb_neg]...]
    list_embeddings -- list, list des embeddings des query, positive, negati
    K -- int, le top K accuracy
    '''
    list_embeddings_query = [list_embeddings[i][0] for i in range(len(list_a
    list_embeddings_pos = [list_embeddings[i][1] for i in range(len(list_all
    list_embeddings_neg = [list_embeddings[i][2] for i in range(len(list_all

    nb_pos = 0
    nb_neg = 0
    for idx in tqdm(range(len(list_embeddings_query)), desc ="Calcul du top_
```

```python
        query = list_embeddings_query[idx]

        similarities_pos = cos_sim(query, list_embeddings_pos).flatten().tol
        similarities_pos = [('pos_{}'.format(i), similarities_pos[i]) for i

        similarities_neg = cos_sim(query, list_embeddings_neg).flatten().tol
        similarities_neg = [('neg_{}'.format(i), similarities_neg[i]) for i

        similarities = similarities_pos+similarities_neg
        similarities = sorted(similarities, key = lambda x: -x[1])
        top_K_ids = [similarities[i][0] for i in range(K)]

        if 'pos_{}'.format(idx) in top_K_ids:
            nb_pos+=1
        if 'neg_{}'.format(idx) in top_K_ids:
            nb_neg+=1
    acc_K_pos = nb_pos/len(list_embeddings)
    acc_K_neg = nb_neg/len(list_embeddings)
    return acc_K_pos, acc_K_neg

acc_K_pos, acc_K_neg = compute_top_K_accuracy_score(list_all_embeddings, K=5
print('acc_K_pos : {} (a maximiser)'.format(acc_K_pos))
print('acc_K_neg : {} (a minimiser)'.format(acc_K_neg))
```

```
Calcul du top_K_accuracy score: 100%|███████████████████████████████████████
█████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:43<00:00, 11.55it/s]
acc_K_pos : 0.9338677354709419 (a maximiser)
acc_K_neg : 0.4168336673346693 (a minimiser)
```

In [ ]:

Etape 4 : Construction d'un modèle en passant par des résumés de section

La démarche ici est la suivante :

- pour chaque section de brevet on réalise un résumé à base des 5 phrases les plus importantes de la section
- on réalise l'embedding de la section par mean pooling des embeddings des phrases résumant la section
- on réalise l'embedding du brevet par mean pooling des embeddings des résumés de section

In [1]:
```
import json
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pickle

from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim
from sentence_transformers.quantization import quantize_embeddings
from sentence_transformers import losses
from sentence_transformers.readers import InputExample
from torch.utils.data import DataLoader
from transformers import AutoTokenizer

import nltk
import numpy as np
from LexRank import degree_centrality_scores
```

Création des embeddings après avoir appliqué des résumés de chacune des sections

Le but ici est d'utiliser un LLM pour créer des résumés de chaque section, puis de créer les embeddings des résumés

In [2]:
```
with open('../data/dataset_patent_sections.json', 'r') as outfile:
    dataset_patent_section = json.load(outfile)
    outfile.close()
```

In [21]:
```
model_name = 'intfloat/e5-small-v2'
model = SentenceTransformer(model_name)
```

In [84]:
```
# Construction du dataset de brevets resumes par section
```

```python
def summarize_vanilla(text, model, max_sentences=5):
    '''
    Fonction pour fournir le resume d'un texte sur la base des ses phrases l
    text -- str, le texte a resumer
    model -- SentenceTransformer, model de calcul des embeddings
    max_sentences -- int, le nombre de phrases max dans le resume
    '''
    sentences = nltk.sent_tokenize(text)
    embeddings = model.encode(sentences)
    similarity_scores = cos_sim(embeddings, embeddings).numpy()
    centrality_scores = degree_centrality_scores(similarity_scores, threshol
    most_central_sentence_indices = np.argsort(-centrality_scores)
    nb_sentences_summary = min(5, len(sentences))
    list_sentences_summary = []
    list_embeddings_summary = []
    for idx in most_central_sentence_indices[0:nb_sentences_summary]:
        list_sentences_summary.append(sentences[idx].strip())
        list_embeddings_summary.append(embeddings[idx])
    summary_embedding = np.mean(list_embeddings_summary, axis=0)
    summary = ' '.join(list_sentences_summary)
    return summary, summary_embedding

dataset_patent_section_summary = {}
for i in tqdm(range(len(dataset_patent_section)), desc ="Construction du dat
    dict_patent = {}
    dict_patent['query'] = dataset_patent_section[str(i)]['query']
    for key in ['pos', 'negative']:
        list_sections = dataset_patent_section[str(i)][key]
        list_sections_summary = []
        for j in range(len(list_sections)):
            text = list_sections[j]['content']
            summary, summary_embedding = summarize_vanilla(text, model, max_
            list_sections_summary.append({'section': list_sections[j]['secti
                                          'content': summary,
                                          'embedding': summary_embedding})
        dict_patent[key] = list_sections_summary
    dataset_patent_section_summary[str(i)] = dict_patent
```

```
Construction du dataset de resumes: 100%|███████████████████████████████████
████████████████████████████████████████████████████████████████████████████
██████████████████| 499/499 [32:49<00:00,  3.95s/it]
```

In [100…

```python
# Construction des embeddings de brevet a partir des embeddings des resumes
dataset_patent_section_embeddings = {}

for i in tqdm(range(len(dataset_patent_section_summary)), desc ="Constructio
    dataset_patent_section_embeddings[str(i)] = {}
    for key in ['pos', 'negative']:
        list_sections_dict = dataset_patent_section_summary[str(i)][key]
        list_embeddings = [list_sections_dict[j]['embedding'] for j in range
        patent_embedding = np.mean([emb for emb in list_embeddings if (not i
        dataset_patent_section_embeddings[str(i)][key] = patent_embedding
```

```
Construction des embeddings de brevets: 100%|███████████████████████████████
████████████████████████████████████████████████████████████████████████████
███████████████| 499/499 [00:00<00:00, 16355.83it/s]
```

In [101…
```
# Ajout des embeddings de query
for i in tqdm(range(len(dataset_patent_section_embeddings)), desc ="Calcul d
    list_sentences = [dataset_patent_section[str(i)]['query']]
    embeddings = model.encode(list_sentences)
    query_embedding = np.mean(embeddings, axis=0)
    dataset_patent_section_embeddings[str(i)]['query'] = query_embedding
```

```
Calcul des embeddings des query: 100%|██████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
██████████████████| 499/499 [00:10<00:00, 47.96it/s]
```

In [102…
```
with open('../data/patent_sections_embeddings_summary_e5-small-v2.pickle', '
    pickle.dump(dataset_patent_section_embeddings, fh)
    fh.close()
```

In [3]:
```
with open('../data/patent_sections_embeddings_summary_e5-small-v2.pickle', '
    dataset_patent_section_embeddings = pickle.load(fh)
    fh.close()
```

In [4]:
```
# Performances -- classification
nb_good_embeddings = 0
for i in range(len(dataset_patent_section_embeddings)):
    embeddings = [
        dataset_patent_section_embeddings[str(i)]['query'],
        dataset_patent_section_embeddings[str(i)]['pos'],
        dataset_patent_section_embeddings[str(i)]['negative']
    ]
    similarities = cos_sim(embeddings[0], embeddings[1:])
    sim_pos, sim_neg = similarities.flatten()
    if sim_pos > sim_neg :
        nb_good_embeddings+=1
perc_good_embeddings = round(100*nb_good_embeddings/len(dataset_patent_secti
print('Embeddings de documents compatibles avec la query: {}, {} %'.format(r
                                                                            p
```

```
Embeddings de documents compatibles avec la query: 408, 81.76 %
```

In [5]:
```
# Performances -- top_K_accuracy
list_all_embeddings = []
for i in range(len(dataset_patent_section_embeddings)):
    emb_query = dataset_patent_section_embeddings[str(i)]['query']
    emb_pos = dataset_patent_section_embeddings[str(i)]['pos']
    emb_neg = dataset_patent_section_embeddings[str(i)]['negative']
    list_all_embeddings.append([emb_query, emb_pos, emb_neg])

def compute_top_K_accuracy_score(list_embeddings, K=5):
    '''
    Fonction pour calculer le top_K_accuracy score a partir d'une liste d'em
    [[emb_query, emb_pos, emb_neg]...]
    list_embeddings -- list, list des embeddings des query, positive, negati
    K -- int, le top K accuracy
    '''
    list_embeddings_query = [list_embeddings[i][0] for i in range(len(list_a
    list_embeddings_pos = [list_embeddings[i][1] for i in range(len(list_all
    list_embeddings_neg = [list_embeddings[i][2] for i in range(len(list_all
```

```
    nb_pos = 0
    nb_neg = 0
    for idx in tqdm(range(len(list_embeddings_query)), desc ="Calcul du top_
        query = list_embeddings_query[idx]

        similarities_pos = cos_sim(query, list_embeddings_pos).flatten().tol
        similarities_pos = [('pos_{}'.format(i), similarities_pos[i]) for i 

        similarities_neg = cos_sim(query, list_embeddings_neg).flatten().tol
        similarities_neg = [('neg_{}'.format(i), similarities_neg[i]) for i 

        similarities = similarities_pos+similarities_neg
        similarities = sorted(similarities, key = lambda x: -x[1])
        top_K_ids = [similarities[i][0] for i in range(K)]

        if 'pos_{}'.format(idx) in top_K_ids:
            nb_pos+=1
        if 'neg_{}'.format(idx) in top_K_ids:
            nb_neg+=1
    acc_K_pos = nb_pos/len(list_embeddings)
    acc_K_neg = nb_neg/len(list_embeddings)
    return acc_K_pos, acc_K_neg

acc_K_pos, acc_K_neg = compute_top_K_accuracy_score(list_all_embeddings, K=5
print('acc_K_pos : {} (a maximiser)'.format(acc_K_pos))
print('acc_K_neg : {} (a minimiser)'.format(acc_K_neg))
```

```
Calcul du top_K_accuracy score: 100%|███████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:44<00:00, 11.24it/s]
acc_K_pos : 0.8837675350701403 (a maximiser)
acc_K_neg : 0.5811623246492986 (a minimiser)
```

In [ ]:

Synthèse et conclusions

Jeu de données

Durant ce test, nous avons exploré un jeu de données composés de contenus de brevet et de leurs résumés. Le jeu de données est construit sous forme de triplet (query, positive_patent, negative_patent) où le positive_patent est sensé être le brevet le plus proche a retrouver a partir de la query, et le negative_patent un brevet proche en terme de contenu, mais qui ne correspond pas a l'objet de la query.

Problème à résoudre

Le problème à résoudre est de trouver une manière de réaliser les embeddings des brevets de telle sorte que les exemples positifs soit plus similaires à la query que les exemples négatifs. De plus, dans une tache de retrieval, pour la query donnee, il faut que l'exemple positif arrive dans le top K, contrairement a l'exemple négatif.

1- Exploration des données

L'exploration des données nous a permis de faire notamment les observations suivantes:

- le jeu de données est composé de contenus de brevet et d'abstracts de brevets
- la longueur des documents est importante, il faut donc réaliser du chunk avant d'ingérer les données dans un LLM

2- Méthodes testées

Nous avons testé trois méthodes pour réaliser l'embedding des brevets, toutes utilisant le modèle intfloat/e5-small-v2 :

- méthode 1: on fait l'embedding directement sur tout le contenu du document (méthode zero-shot). On teste aussi la variante en finetunant le modèle avec la contrastive loss.
- méthode 2 : on fait l'embedding des brevets a partir des embeddings de sections en realisant un max pooling. Chaque embedding de section etant lui-même obtenu par max pooling des embeddings de chunks composant la section
- méthode 3 : on fait l'embedding des brevets a partir des embeddings de résumés de sections en réalisant un max pooling. Les résumés de section étant constitué des 5 phrases les plus importantes constituant la section

3- résultats obtenus

- méthode 1 Zero-Shot:
  - Accuracy : 74.15 %
  - top_5_accuracy positive : 93.4 %

- top_5_accuracy negative : 71.3 %
- méthode 1 Finetuning (probleme d'overfitting) :
  - Accuracy : 100 %
  - top_5_accuracy positive : 19.4 %
  - top_5_accuracy negative : 0 %
- méthode 2:
  - Accuracy : 90.18 %
  - top_5_accuracy positive : 93.4 %
  - top_5_accuracy negative : 41.7 %
- méthode 3:
  - Accuracy : 81.76 %
  - top_5_accuracy positive : 88.37 %
  - top_5_accuracy negative : 58.1 %

4- Interprétation des résultats

La méthode 2 donne les meilleures résultats. Elle a nécessité un travail plus approfondi pour comprendre les brevets en exploitant les sections.

Les résultats de la méthode 3 ne sont pas si satisfaisants car la manière de générer les résumés n'est efficace. Il faudrait plutôt générer des vrais résumés à partir de LLM entrainés sur une tâche de summarization, plutôt que de prendre directement les phrases du document.

5- Idées a explorer pour la suite

Les idées qui viennent tout de suite pour une future exploration sont les suivantes :

- Tester des LLM plus performants
- Résumer chaque section directement par un LLM
- Faire le finetuning des modèles en exploitant la contrastive loss sur les chunks construits avec la méthode 2 plutôt qu'en brute force directe sur tout le document
- Avoir un plus gros dataset, avec plus d'exemples négatifs pour chaque query

In [ ]: