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)
```

```python
    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]:
```python
# 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]:
```python
# 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 [ ]: