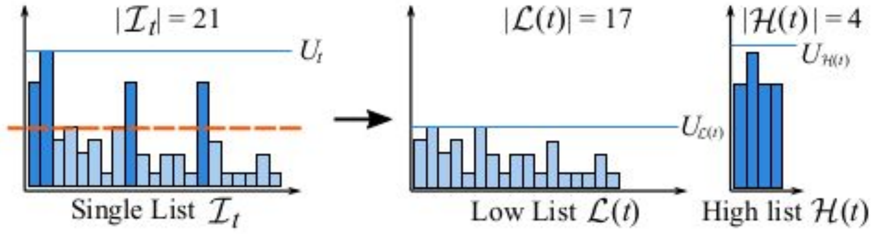
“Accelerating Sparse Indexes Via Term Impact Decomposition”

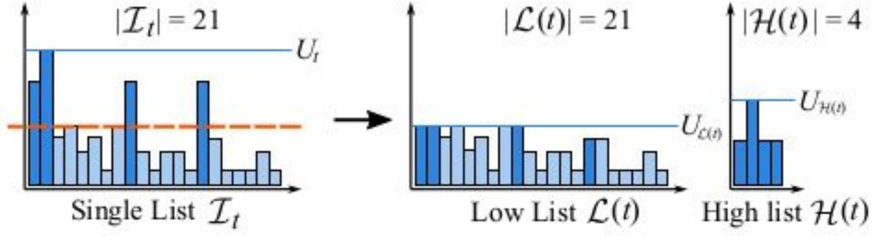
Joel Mackenzie, Antonio Mallia, Alistair Moffat et Matthias Petri

Nour BOUCHOUCHI
Nolwenn PIGEON

Deux techniques pour améliorer l'efficiency : list splitting et postings clipping

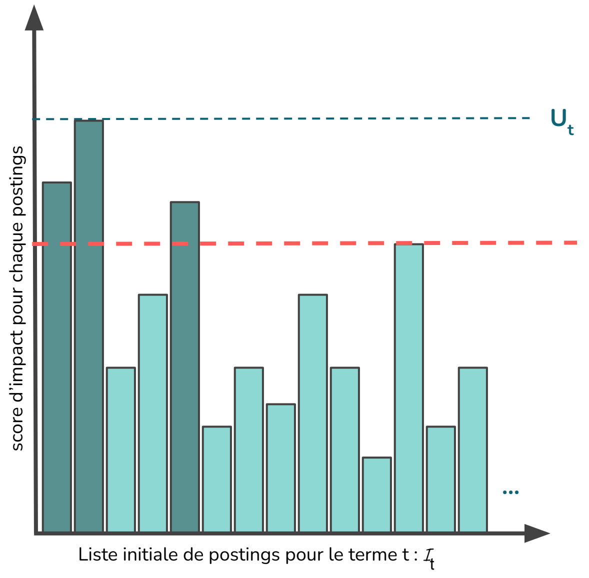


(a) List Splitting



(b) Postings Clipping

List splitting

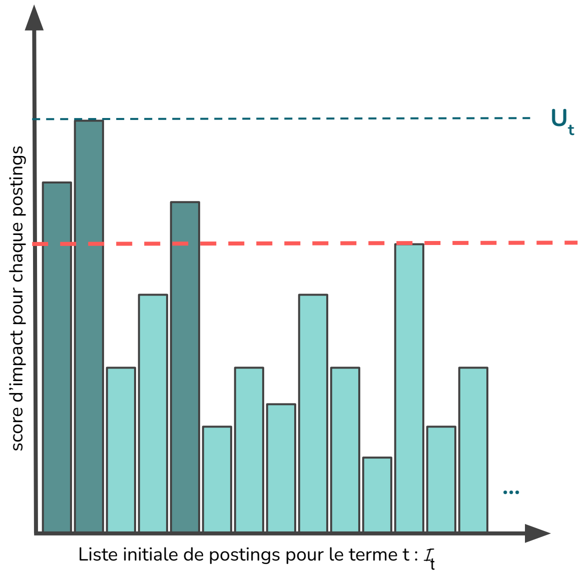


1/64e des postings ont un score d'impact supérieur

List splitting

$|\mathcal{I}_{\text{project}}| = 9\ 142$

$|\mathcal{I}_{\text{project}}| = 10\ 226$



U_t
score d'impact pour chaque postings
...
Liste initiale de postings pour le terme t : $\mathcal{I}_t$

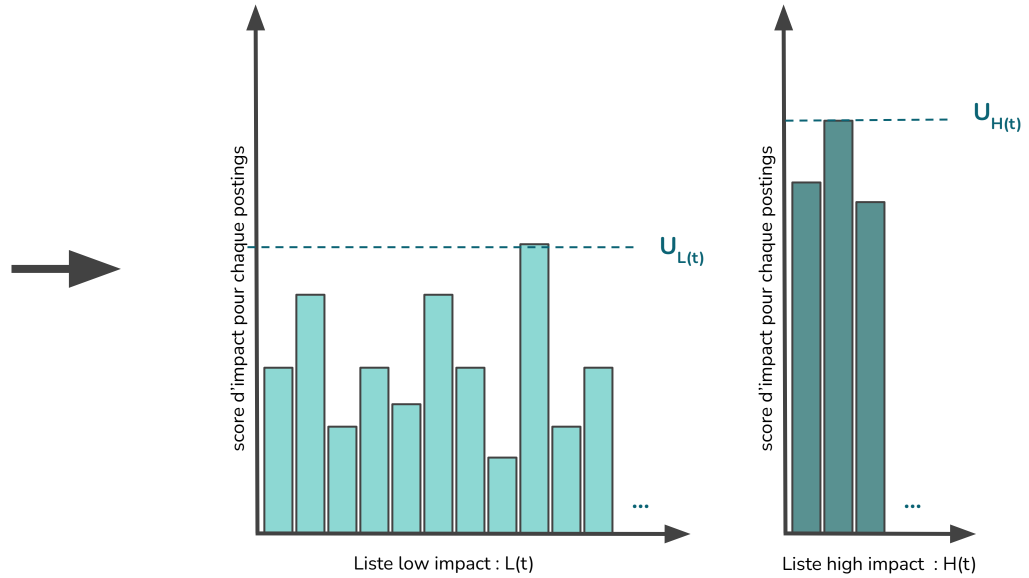
DeepImpact : 1,1 Go
BM25 : 416 MO

$|\mathcal{L}(projet)| = 9\ 000$

$|\mathcal{L}(project)| = 10\ 117$

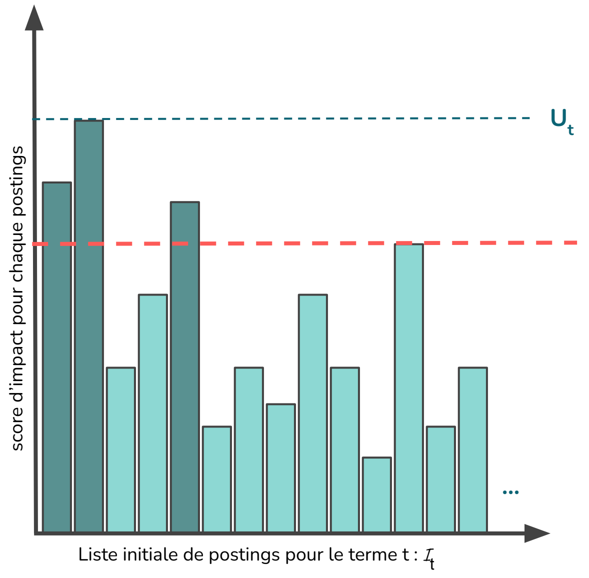
$|\mathcal{H}(projet)| = 142$

$|\mathcal{H}(project)| = 109$



DeepImpact : 1,1 Go
BM25 : 420 MO

Postings clipping

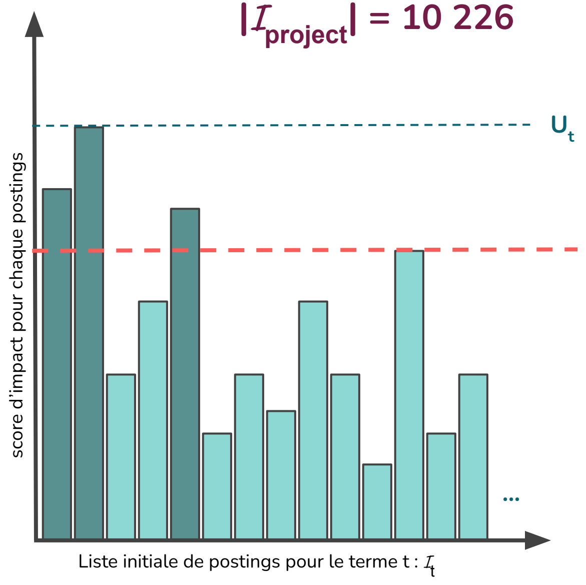


1/64e des postings ont un score d'impact supérieur

Postings clipping

$|\mathcal{I}_{\text{project}}| = 9\ 142$

$|\mathcal{I}_{\text{project}}| = 10\ 226$



U_t
score d'impact pour chaque postings
...
Liste initiale de postings pour le terme t : $\mathcal{I}_t$

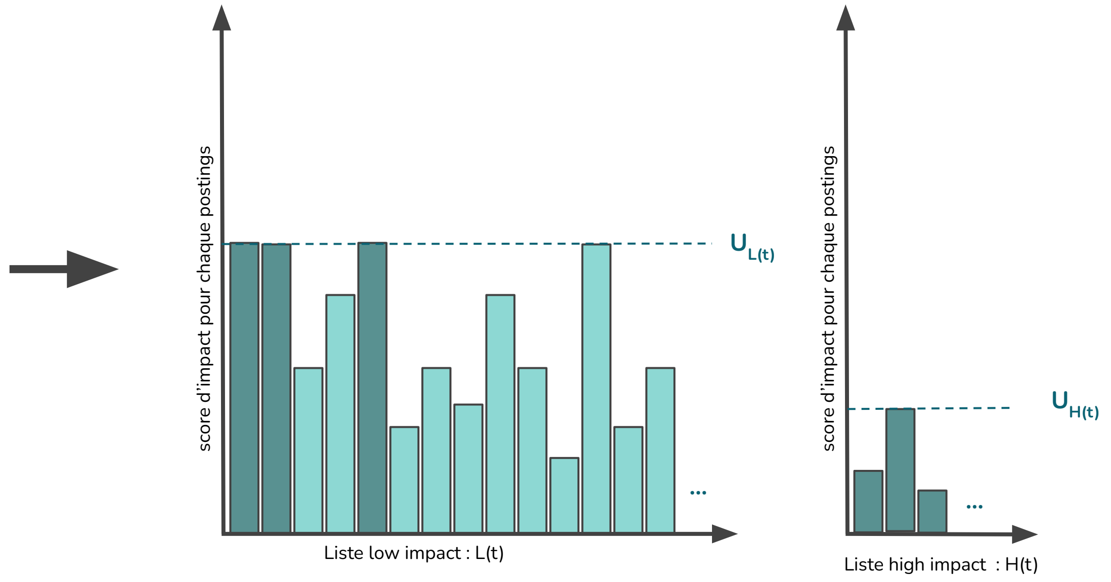
DeepImpact : 1,1 Go
BM25 : 420 M0

$|\mathcal{L}(projet)| = 9\ 142$

$|\mathcal{L}(projet)| = 10\ 226$

$|\mathcal{H}(projet)| = 142$

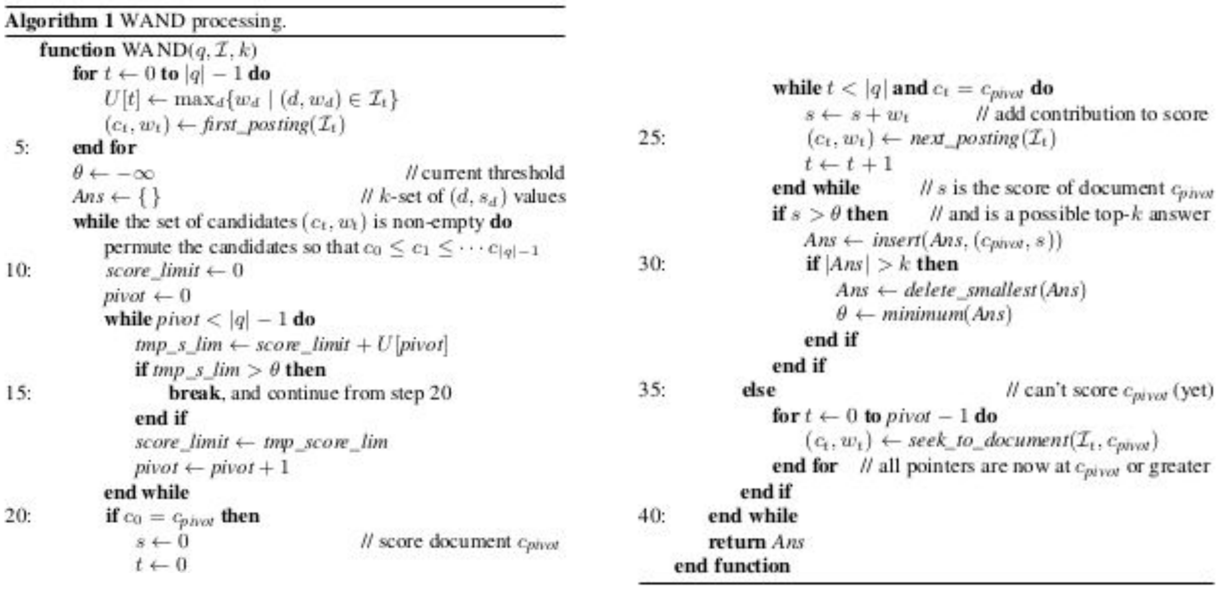
$|\mathcal{H}(projet)| = 109$



DeepImpact : 1,2 Go
BM25 : 424 M0

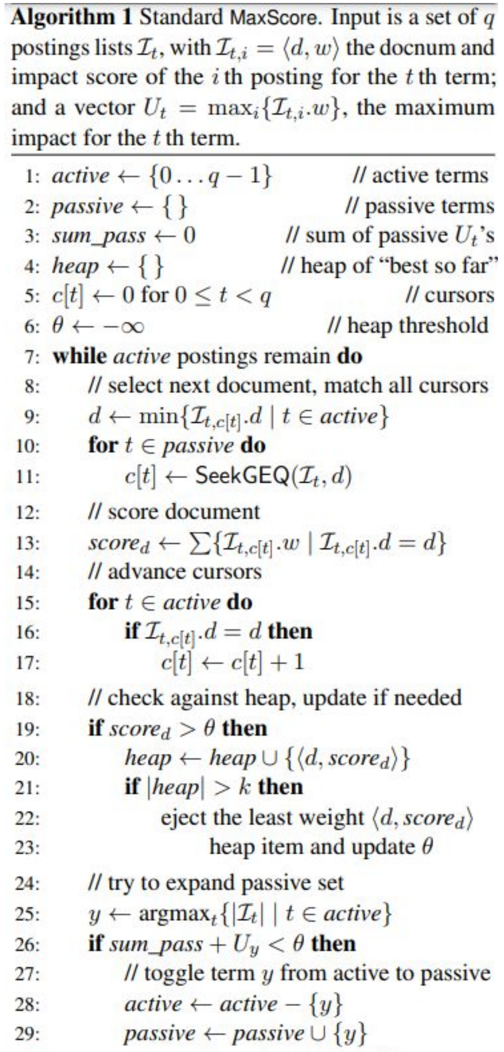
Algorithme Wand et MaxScore

Smart Bounds : Prise en compte de UL(t) et UH(t).
Mise à jour de la borne supérieure devient : UH(t) - UL(t).



Algorithm 1 WAND processing.

```
    function WAND($q, \mathcal{I}, k$)
        for $t \leftarrow 0$ to $|q| - 1$ do
            $U[t] \leftarrow \max_d\{w_d \mid (d, w_d) \in \mathcal{I}_t\}$
            $(c_t, w_t) \leftarrow first\_posting(\mathcal{I}_t)$
 5:     end for
        $\theta \leftarrow -\infty$                          // current threshold
        $Ans \leftarrow \{\}$                         // $k$-set of $(d, s_d)$ values
        while the set of candidates $(c_t, w_t)$ is non-empty do
            permute the candidates so that $c_0 \leq c_1 \leq \cdots c_{|q|-1}$
10:         $score\_limit \leftarrow 0$
            $pivot \leftarrow 0$
            while $pivot < |q| - 1$ do
                $tmp\_s\_lim \leftarrow score\_limit + U[pivot]$
                if $tmp\_s\_lim > \theta$ then
15:                 break, and continue from step 20
                end if
                $score\_limit \leftarrow tmp\_score\_lim$
                $pivot \leftarrow pivot + 1$
            end while
20:         if $c_0 = c_{pivot}$ then
                $s \leftarrow 0$                        // score document $c_{pivot}$
                $t \leftarrow 0$
                while $t < |q|$ and $c_t = c_{pivot}$ do
                    $s \leftarrow s + w_t$          // add contribution to score
25:                 $(c_t, w_t) \leftarrow next\_posting(\mathcal{I}_t)$
                    $t \leftarrow t + 1$
                end while        // $s$ is the score of document $c_{pivot}$
                if $s > \theta$ then      // and is a possible top-$k$ answer
                    $Ans \leftarrow insert(Ans, (c_{pivot}, s))$
30:                 if $|Ans| > k$ then
                        $Ans \leftarrow delete\_smallest(Ans)$
                        $\theta \leftarrow minimum(Ans)$
                    end if
                end if
35:         else                               // can't score $c_{pivot}$ (yet)
                for $t \leftarrow 0$ to $pivot - 1$ do
                    $(c_t, w_t) \leftarrow seek\_to\_document(\mathcal{I}_t, c_{pivot})$
                end for    // all pointers are now at $c_{pivot}$ or greater
            end if
40:     end while
        return $Ans$
    end function
```

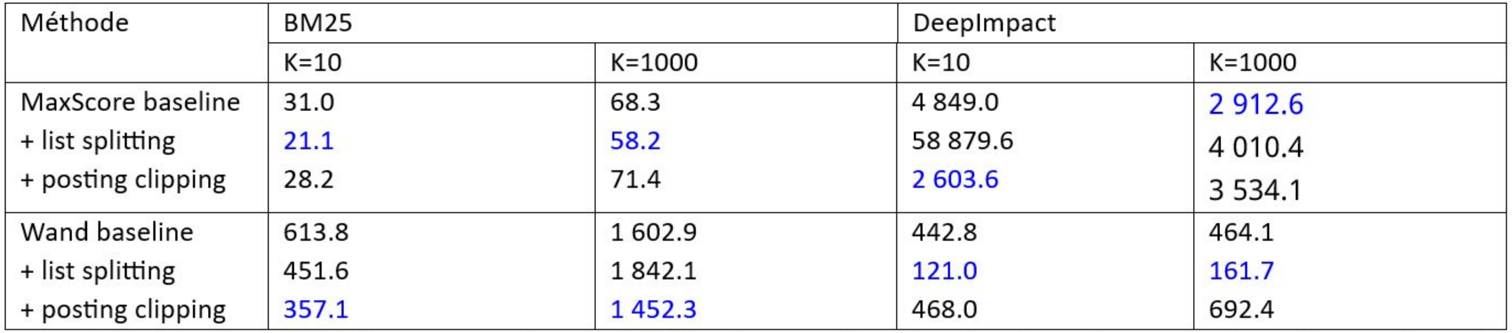


Algorithm 1 Standard MaxScore. Input is a set of q postings lists $\mathcal{I}_t$, with $\mathcal{I}_{t,i} = \langle d, w \rangle$ the docnum and impact score of the i th posting for the t th term; and a vector $U_t = \max_i\{\mathcal{I}_{t,i}.w\}$, the maximum impact for the t th term.

```
 1: $active \leftarrow \{0 \ldots q-1\}$            // active terms
 2: $passive \leftarrow \{\}$                        // passive terms
 3: $sum\_pass \leftarrow 0$                 // sum of passive $U_t$'s
 4: $heap \leftarrow \{\}$                 // heap of "best so far"
 5: $c[t] \leftarrow 0$ for $0 \leq t < q$                    // cursors
 6: $\theta \leftarrow -\infty$                         // heap threshold
 7: while $active$ postings remain do
 8:     // select next document, match all cursors
 9:     $d \leftarrow \min\{\mathcal{I}_{t,c[t]}.d \mid t \in active\}$
10:     for $t \in passive$ do
11:         $c[t] \leftarrow$ SeekGEQ$(\mathcal{I}_t, d)$
12:     // score document
13:     $score_d \leftarrow \sum\{\mathcal{I}_{t,c[t]}.w \mid \mathcal{I}_{t,c[t]}.d = d\}$
14:     // advance cursors
15:     for $t \in active$ do
16:         if $\mathcal{I}_{t,c[t]}.d = d$ then
17:             $c[t] \leftarrow c[t] + 1$
18:     // check against heap, update if needed
19:     if $score_d > \theta$ then
20:         $heap \leftarrow heap \cup \{\langle d, score_d \rangle\}$
21:         if $|heap| > k$ then
22:             eject the least weight $\langle d, score_d \rangle$
23:                 heap item and update $\theta$
24:     // try to expand passive set
25:     $y \leftarrow \text{argmax}_t\{|\mathcal{I}_t| \mid t \in active\}$
26:     if $sum\_pass + U_y < \theta$ then
27:         // toggle term $y$ from active to passive
28:         $active \leftarrow active - \{y\}$
29:         $passive \leftarrow passive \cup \{y\}$
```

Résultats

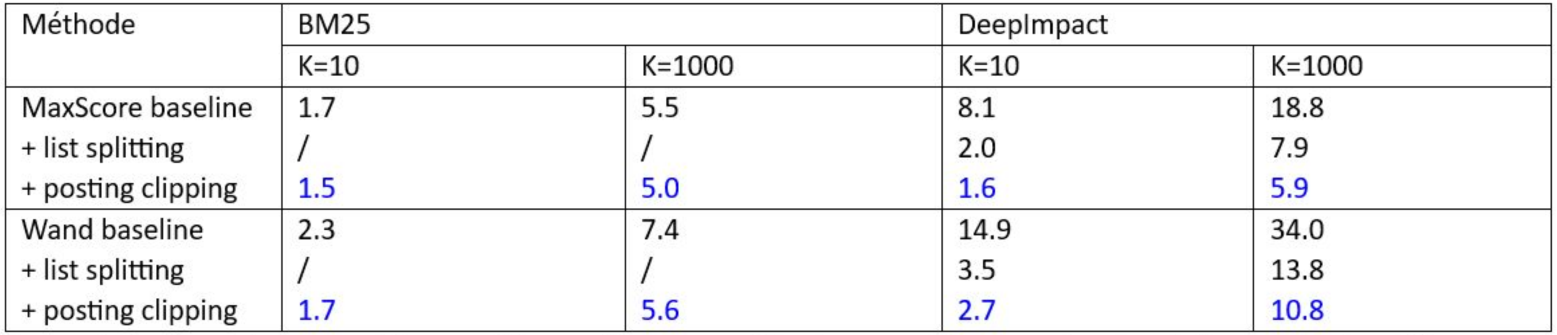
Nos résultats :



Méthode	BM25		DeepImpact	
	K=10	K=1000	K=10	K=1000
MaxScore baseline	31.0	68.3	4 849.0	2 912.6
+ list splitting	21.1	58.2	58 879.6	4 010.4
+ posting clipping	28.2	71.4	2 603.6	3 534.1
Wand baseline	613.8	1 602.9	442.8	464.1
+ list splitting	451.6	1 842.1	121.0	161.7
+ posting clipping	357.1	1 452.3	468.0	692.4

Temps de traitement des requêtes en secondes par requête, deux modèles de recherche et deux approches d’élagage dynamique

Résultats de l’article :



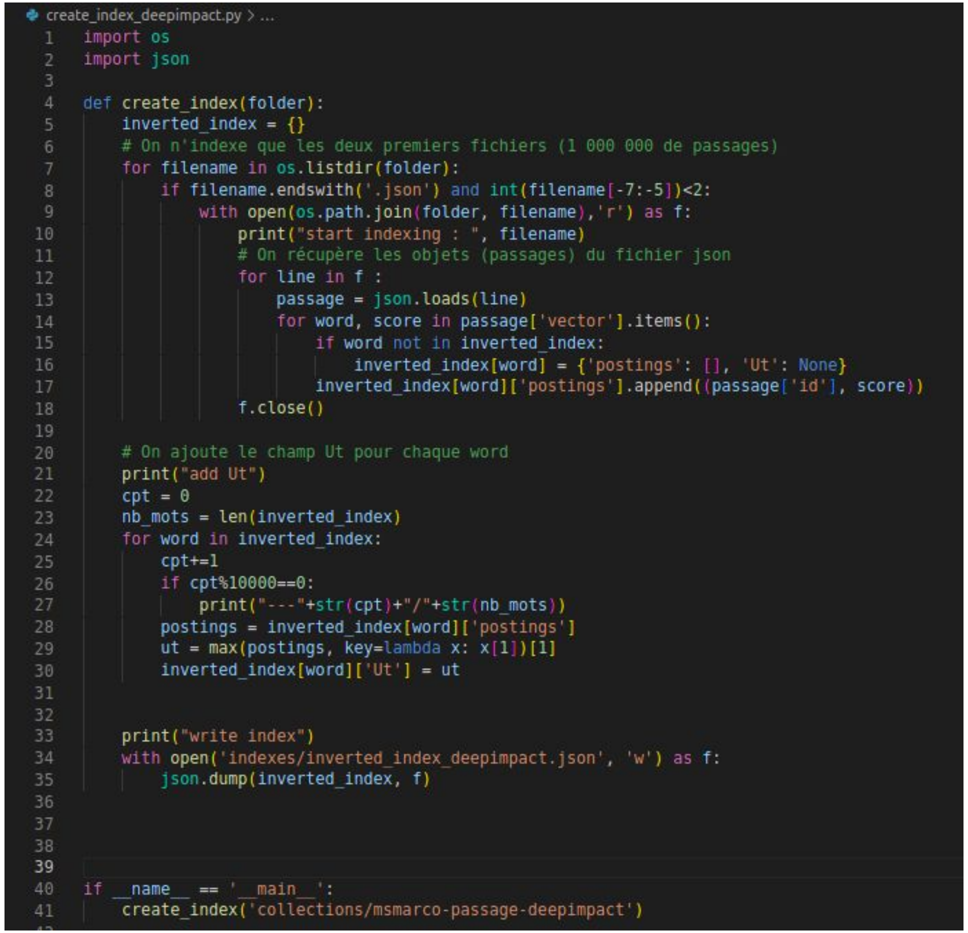
Méthode	BM25		DeepImpact	
	K=10	K=1000	K=10	K=1000
MaxScore baseline	1.7	5.5	8.1	18.8
+ list splitting	/	/	2.0	7.9
+ posting clipping	1.5	5.0	1.6	5.9
Wand baseline	2.3	7.4	14.9	34.0
+ list splitting	/	/	3.5	13.8
+ posting clipping	1.7	5.6	2.7	10.8

Temps de traitement des requêtes en millisecondes par requête, deux modèles de recherche et deux approches d'élagage dynamique

Annexes

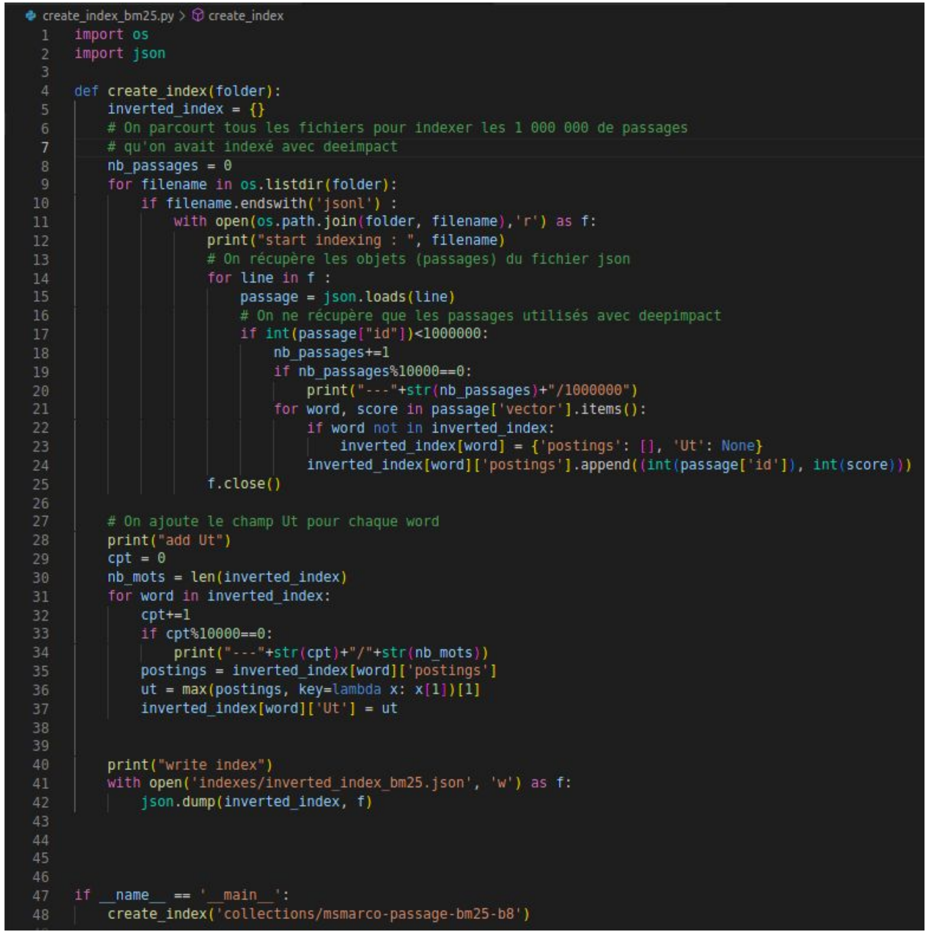
Structure du code

- **create_index_deepimpact.py**
  - Permet de créer un index inversé à partir des index MSMARCOv1 avec Deepimpact et de l’enregistrer en mémoire sous format json. On indexe les 1 million premiers passages pour des raisons de problème de passage à l’échelle.
- **create_index_bm25.py**
  - Permet de créer un index inversé à partir des index MSMARCOv1 avec BM25 et de l’enregistrer en mémoire sous format json. On fait attention à conserver les mêmes passages que ceux qui ont été sélectionnés pour DeepImpact.
- **postings_clipping.py**
  - Permet de créer les index avec postings clipping pour DeepImpact et BM25 à partir des index inversés préalablement créés et de les enregistrer en mémoire sous format json.
- **list_splitting.py**
  - Permet de créer les index avec list splitting pour DeepImpact et BM25 à partir des index inversés préalablement créés et de les enregistrer en mémoire sous format json.
- **wand.py**
  - Permet de calculer le temps moyen des requêtes à l’aide de WAND, pour k=10 et k=1000, à partir des index inversés, avec postings clipping et avec list splitting.
- **maxscore.py**
  - Permet de calculer le temps moyen des requêtes à l’aide de MaxScore pour k=10 et k=1000, à partir des index inversés, avec postings clipping et avec list splitting.


create_index_deepimpact.py > ...

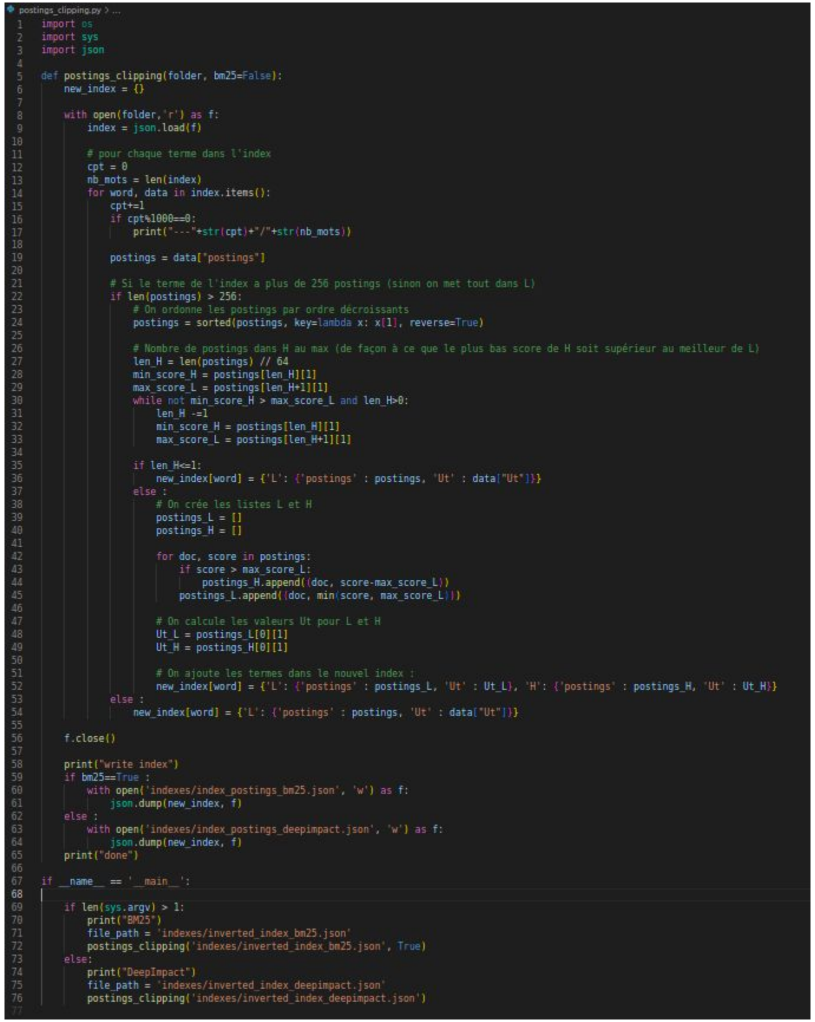
```python
import os
import json

def create_index(folder):
    inverted_index = {}
    # On n'indexe que les deux premiers fichiers (1 000 000 de passages)
    for filename in os.listdir(folder):
        if filename.endswith('.json') and int(filename[-7:-5])<2:
            with open(os.path.join(folder, filename),'r') as f:
                print("start indexing : ", filename)
                # On récupère les objets (passages) du fichier json
                for line in f :
                    passage = json.loads(line)
                    for word, score in passage['vector'].items():
                        if word not in inverted_index:
                            inverted_index[word] = {'postings': [], 'Ut': None}
                        inverted_index[word]['postings'].append((passage['id'], score))
                f.close()

    # On ajoute le champ Ut pour chaque word
    print("add Ut")
    cpt = 0
    nb_mots = len(inverted_index)
    for word in inverted_index:
        cpt+=1
        if cpt%10000==0:
            print("---"+str(cpt)+"/"+str(nb_mots))
        postings = inverted_index[word]['postings']
        ut = max(postings, key=lambda x: x[1])[1]
        inverted_index[word]['Ut'] = ut


    print("write index")
    with open('indexes/inverted_index_deepimpact.json', 'w') as f:
        json.dump(inverted_index, f)




if __name__ == '__main__':
    create_index('collections/msmarco-passage-deepimpact')
```

create_index_deepimpact.py


create_index_bm25.py > create_index

```python
import os
import json

def create_index(folder):
    inverted_index = {}
    # On parcourt tous les fichiers pour indexer les 1 000 000 de passages
    # qu'on avait indexé avec deeimpact
    nb_passages = 0
    for filename in os.listdir(folder):
        if filename.endswith('jsonl') :
            with open(os.path.join(folder, filename),'r') as f:
                print("start indexing : ", filename)
                # On récupère les objets (passages) du fichier json
                for line in f :
                    passage = json.loads(line)
                    # On ne récupère que les passages utilisés avec deepimpact
                    if int(passage["id"])<1000000:
                        nb_passages+=1
                        if nb_passages%10000==0:
                            print("---"+str(nb_passages)+"/1000000")
                        for word, score in passage['vector'].items():
                            if word not in inverted_index:
                                inverted_index[word] = {'postings': [], 'Ut': None}
                            inverted_index[word]['postings'].append((int(passage['id']), int(score)))
                f.close()

    # On ajoute le champ Ut pour chaque word
    print("add Ut")
    cpt = 0
    nb_mots = len(inverted_index)
    for word in inverted_index:
        cpt+=1
        if cpt%10000==0:
            print("---"+str(cpt)+"/"+str(nb_mots))
        postings = inverted_index[word]['postings']
        ut = max(postings, key=lambda x: x[1])[1]
        inverted_index[word]['Ut'] = ut


    print("write index")
    with open('indexes/inverted_index_bm25.json', 'w') as f:
        json.dump(inverted_index, f)




if __name__ == '__main__':
    create_index('collections/msmarco-passage-bm25-b8')
```

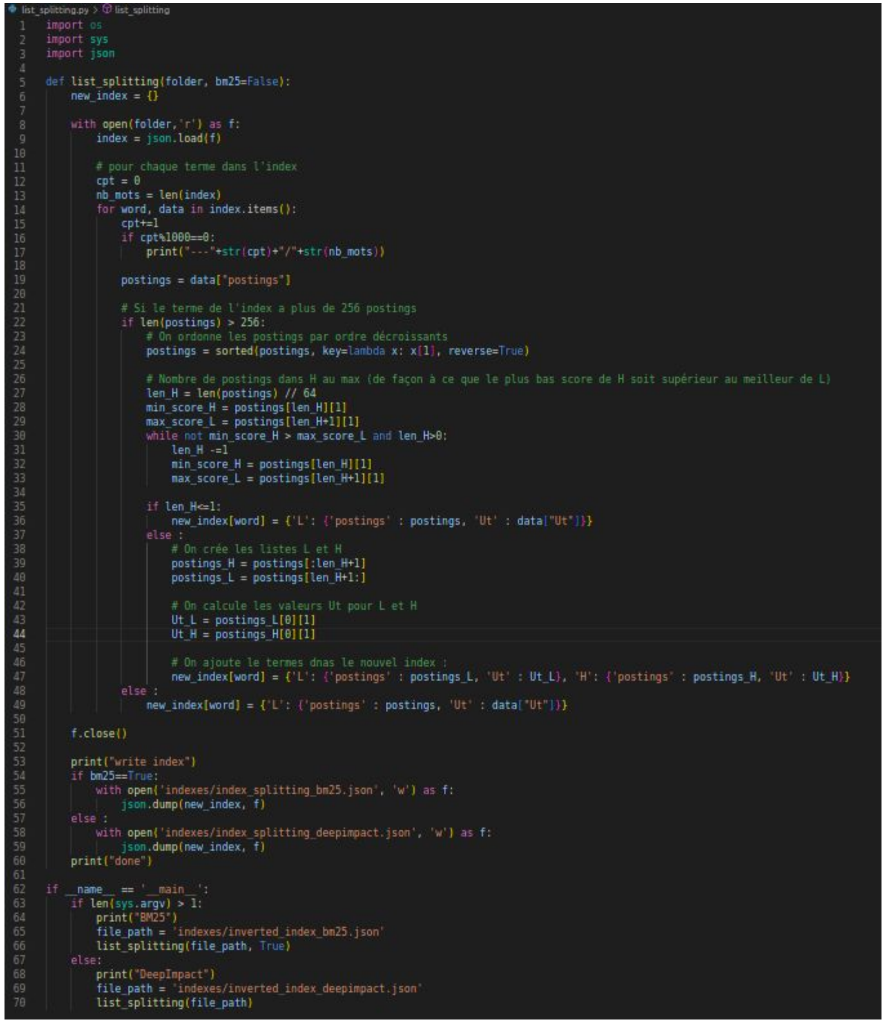
create_index_bm25.py



postings_clipping.py > ...

```python
import os
import sys
import json

def postings_clipping(folder, bm25=False):
    new_index = {}

    with open(folder,'r') as f:
        index = json.load(f)

        # pour chaque terme dans l'index
        cpt = 0
        nb_mots = len(index)
        for word, data in index.items():
            cpt+=1
            if cpt%1000==0:
                print("---"+str(cpt)+"/"+str(nb_mots))

            postings = data["postings"]

            # Si le terme de l'index a plus de 256 postings (sinon on met tout dans L)
            if len(postings) > 256:
                # On ordonne les postings par ordre décroissants
                postings = sorted(postings, key=lambda x: x[1], reverse=True)

                # Nombre de postings dans H au max (de façon à ce que le plus bas score de H soit supérieur au meilleur de L)
                len_H = len(postings) // 64
                min_score_H = postings[len_H][1]
                max_score_L = postings[len_H+1][1]
                while not min_score_H > max_score_L and len_H>0:
                    len_H -=1
                    min_score_H = postings[len_H][1]
                    max_score_L = postings[len_H+1][1]

                if len_H<=1:
                    new_index[word] = {'L': {'postings' : postings, 'Ut' : data["Ut"]}}
                else :
                    # On crée les listes L et H
                    postings_L = []
                    postings_H = []

                    for doc, score in postings:
                        if score > max_score_L:
                            postings_H.append((doc, score-max_score_L))
                        postings_L.append((doc, min(score, max_score_L)))

                    # On calcule les valeurs Ut pour L et H
                    Ut_L = postings_L[0][1]
                    Ut_H = postings_H[0][1]

                    # On ajoute les termes dans le nouvel index :
                    new_index[word] = {'L': {'postings' : postings_L, 'Ut' : Ut_L}, 'H': {'postings' : postings_H, 'Ut' : Ut_H}}
            else :
                new_index[word] = {'L': {'postings' : postings, 'Ut' : data["Ut"]}}

    f.close()

    print("write index")
    if bm25==True :
        with open('indexes/index_postings_bm25.json', 'w') as f:
            json.dump(new_index, f)
    else :
        with open('indexes/index_postings_deepimpact.json', 'w') as f:
            json.dump(new_index, f)
    print("done")

if __name__ == '__main__':

    if len(sys.argv) > 1:
        print("BM25")
        file_path = 'indexes/inverted_index_bm25.json'
        postings_clipping('indexes/inverted_index_bm25.json', True)
    else:
        print("DeepImpact")
        file_path = 'indexes/inverted_index_deepimpact.json'
        postings_clipping('indexes/inverted_index_deepimpact.json')
```

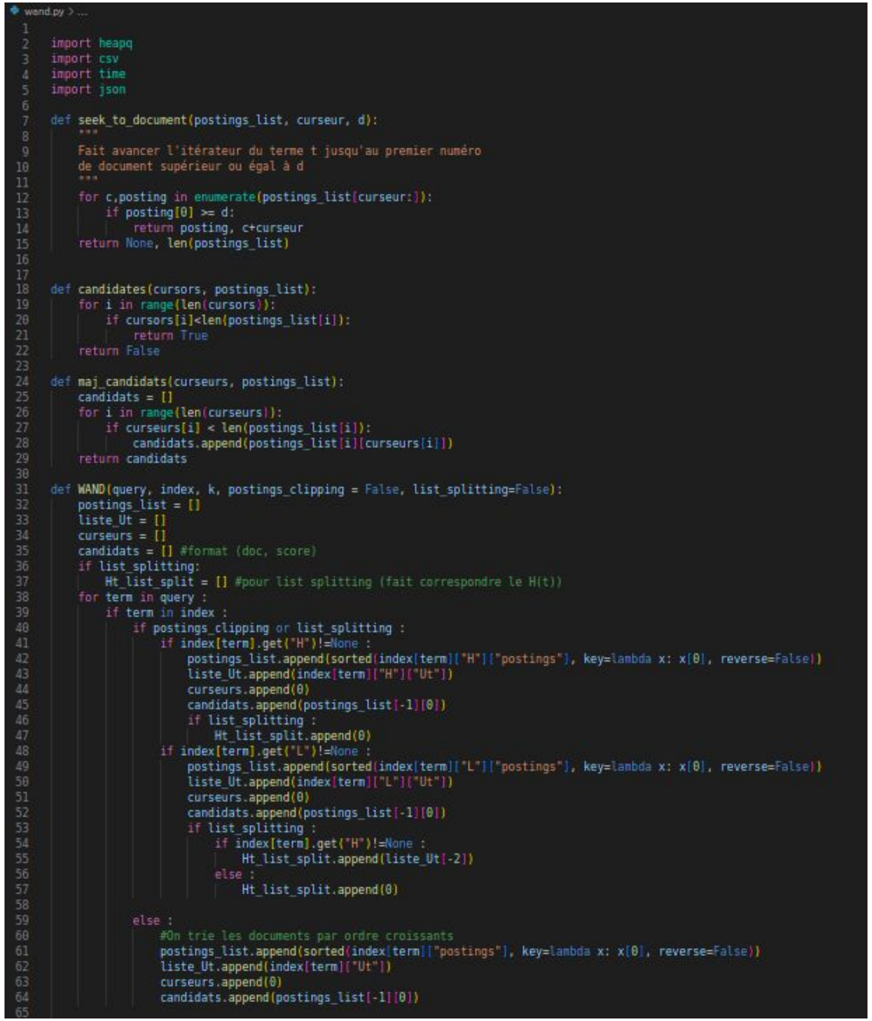
postings_clipping.py



list_splitting.py > list_splitting

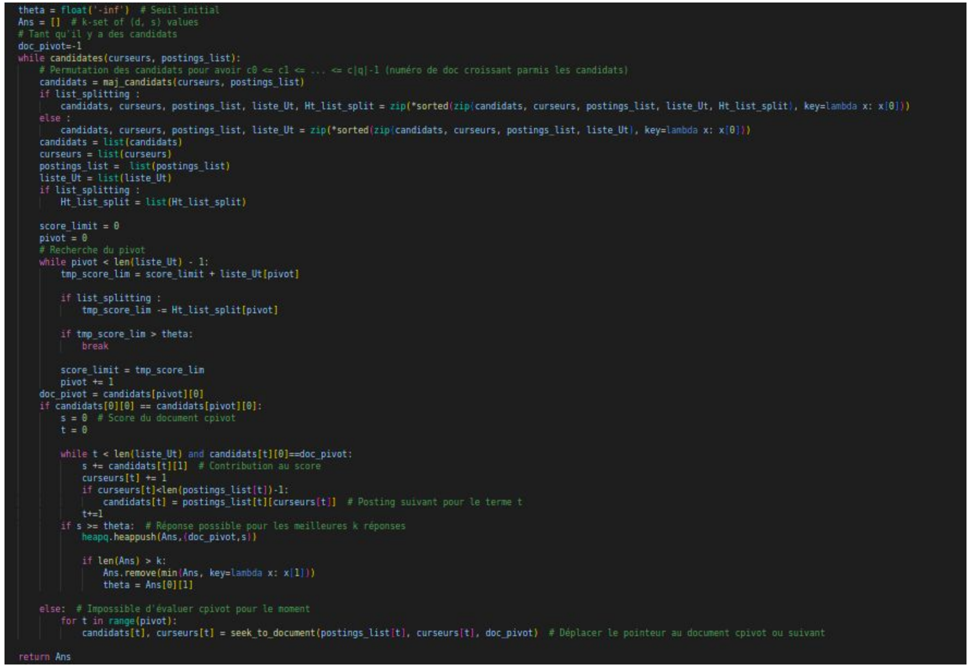
```python
import os
import sys
import json

def list_splitting(folder, bm25=False):
    new_index = {}

    with open(folder,'r') as f:
        index = json.load(f)

        # pour chaque terme dans l'index
        cpt = 0
        nb_mots = len(index)
        for word, data in index.items():
            cpt+=1
            if cpt%1000==0:
                print("---"+str(cpt)+"/"+str(nb_mots))

            postings = data["postings"]

            # Si le terme de l'index a plus de 256 postings
            if len(postings) > 256:
                # On ordonne les postings par ordre décroissants
                postings = sorted(postings, key=lambda x: x[1], reverse=True)

                # Nombre de postings dans H au max (de façon à ce que le plus bas score de H soit supérieur au meilleur de L)
                len_H = len(postings) // 64
                min_score_H = postings[len_H][1]
                max_score_L = postings[len_H+1][1]
                while not min_score_H > max_score_L and len_H>0:
                    len_H -=1
                    min_score_H = postings[len_H][1]
                    max_score_L = postings[len_H+1][1]

                if len_H<=1:
                    new_index[word] = {'L': {'postings' : postings, 'Ut' : data["Ut"]}}
                else :
                    # On crée les listes L et H
                    postings_H = postings[:len_H+1]
                    postings_L = postings[len_H+1:]

                    # On calcule les valeurs Ut pour L et H
                    Ut_L = postings_L[0][1]
                    Ut_H = postings_H[0][1]

                    # On ajoute le termes dnas le nouvel index :
                    new_index[word] = {'L': {'postings' : postings_L, 'Ut' : Ut_L}, 'H': {'postings' : postings_H, 'Ut' : Ut_H}}
            else :
                new_index[word] = {'L': {'postings' : postings, 'Ut' : data["Ut"]}}

    f.close()

    print("write index")
    if bm25==True:
        with open('indexes/index_splitting_bm25.json', 'w') as f:
            json.dump(new_index, f)
    else :
        with open('indexes/index_splitting_deepimpact.json', 'w') as f:
            json.dump(new_index, f)
    print("done")

if __name__ == '__main__':
    if len(sys.argv) > 1:
        print("BM25")
        file_path = 'indexes/inverted_index_bm25.json'
        list_splitting(file_path, True)
    else:
        print("DeepImpact")
        file_path = 'indexes/inverted_index_deepimpact.json'
        list_splitting(file_path)
```

list_splitting.py



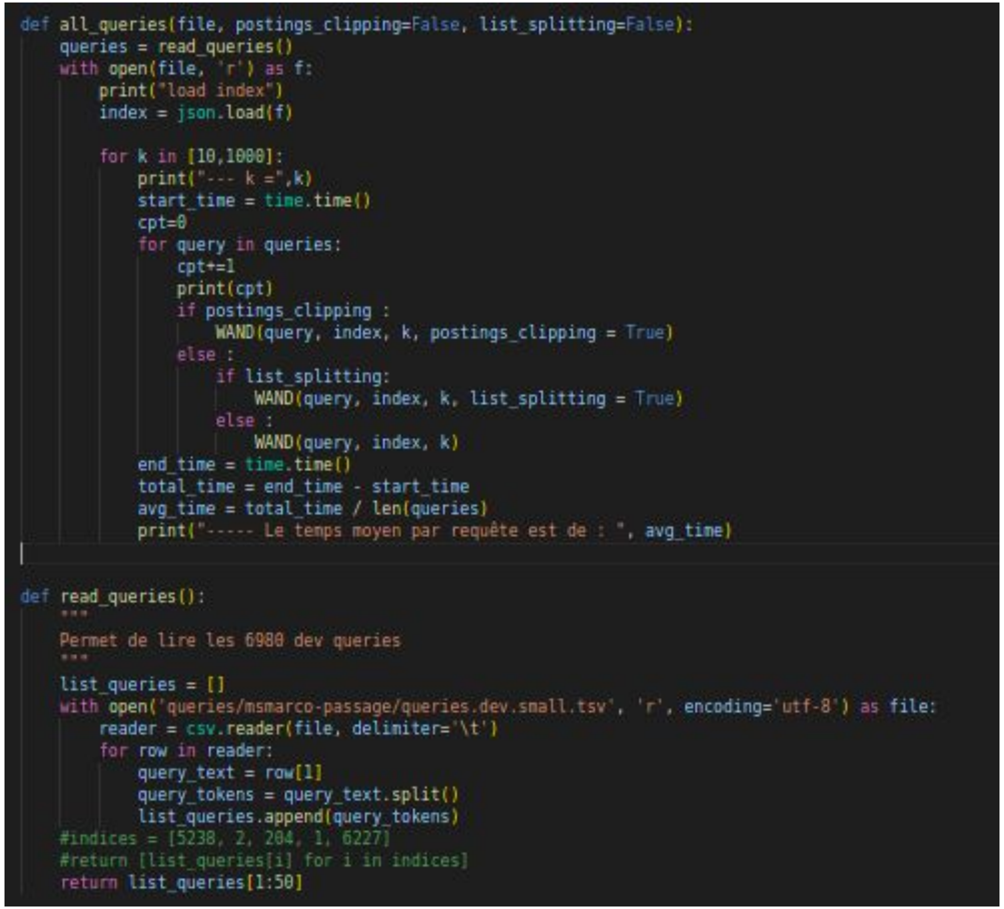
wand.py > ...

```python

import heapq
import csv
import time
import json

def seek_to_document(postings_list, curseur, d):
    """
    Fait avancer l'itérateur du terme t jusqu'au premier numéro
    de document supérieur ou égal à d
    """
    for c,posting in enumerate(postings_list[curseur:]):
        if posting[0] >= d:
            return posting, c+curseur
    return None, len(postings_list)


def candidates(cursors, postings_list):
    for i in range(len(cursors)):
        if cursors[i]<len(postings_list[i]):
            return True
    return False

def maj_candidats(curseurs, postings_list):
    candidats = []
    for i in range(len(curseurs)):
        if curseurs[i] < len(postings_list[i]):
            candidats.append(postings_list[i][curseurs[i]])
    return candidats

def WAND(query, index, k, postings_clipping = False, list_splitting=False):
    postings_list = []
    liste_Ut = []
    curseurs = []
    candidats = [] #format (doc, score)
    if list_splitting:
        Ht_list_split = [] #pour list splitting (fait correspondre le H(t))
    for term in query :
        if term in index :
            if postings_clipping or list_splitting :
                if index[term].get("H")!=None :
                    postings_list.append(sorted(index[term]["H"]["postings"], key=lambda x: x[0], reverse=False))
                    liste_Ut.append(index[term]["H"]["Ut"])
                    curseurs.append(0)
                    candidats.append(postings_list[-1][0])
                    if list_splitting :
                        Ht_list_split.append(0)
                if index[term].get("L")!=None :
                    postings_list.append(sorted(index[term]["L"]["postings"], key=lambda x: x[0], reverse=False))
                    liste_Ut.append(index[term]["L"]["Ut"])
                    curseurs.append(0)
                    candidats.append(postings_list[-1][0])
                    if list_splitting :
                        if index[term].get("H")!=None :
                            Ht_list_split.append(liste_Ut[-2])
                        else :
                            Ht_list_split.append(0)

            else :
                #On trie les documents par ordre croissants
                postings_list.append(sorted(index[term]["postings"], key=lambda x: x[0], reverse=False))
                liste_Ut.append(index[term]["Ut"])
                curseurs.append(0)
                candidats.append(postings_list[-1][0])

```

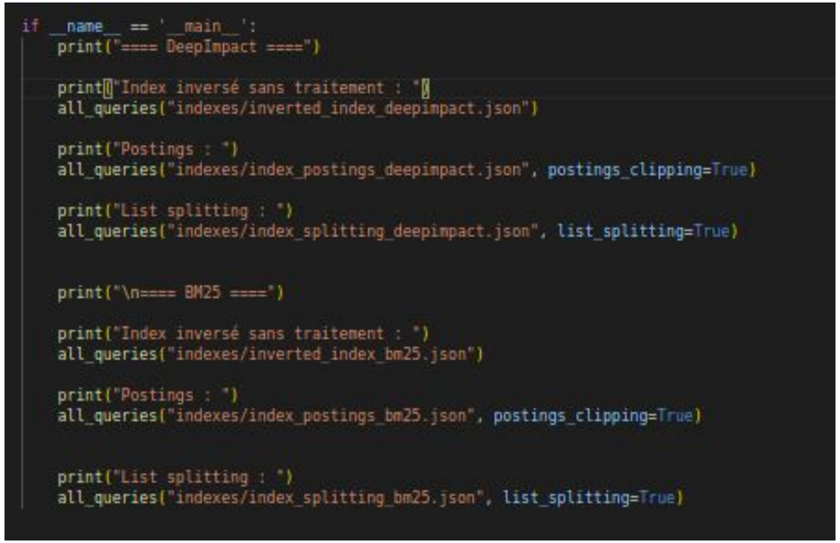


```python
theta = float('-inf')  # Seuil initial
Ans = []  # k-set of (d, s) values
# Tant qu'il y a des candidats
doc_pivot=-1
while candidates(curseurs, postings_list):
    # Permutation des candidats pour avoir c0 <= c1 <= ... <= c|q|-1 (numéro de doc croissant parmis les candidats)
    candidats = maj_candidats(curseurs, postings_list)
    if list_splitting :
        candidats, curseurs, postings_list, liste_Ut, Ht_list_split = zip(*sorted(zip(candidats, curseurs, postings_list, liste_Ut, Ht_list_split), key=lambda x: x[0]))
    else :
        candidats, curseurs, postings_list, liste_Ut = zip(*sorted(zip(candidats, curseurs, postings_list, liste_Ut), key=lambda x: x[0]))
    candidats = list(candidats)
    curseurs = list(curseurs)
    postings_list =  list(postings_list)
    liste_Ut = list(liste_Ut)
    if list_splitting :
        Ht_list_split = list(Ht_list_split)

    score_limit = 0
    pivot = 0
    # Recherche du pivot
    while pivot < len(liste_Ut) - 1:
        tmp_score_lim = score_limit + liste_Ut[pivot]

        if list_splitting :
            tmp_score_lim -= Ht_list_split[pivot]

        if tmp_score_lim > theta:
            break

        score_limit = tmp_score_lim
        pivot += 1
    doc_pivot = candidats[pivot][0]
    if candidats[0][0] == candidats[pivot][0]:
        s = 0  # Score du document cpivot
        t = 0

        while t < len(liste_Ut) and candidats[t][0]==doc_pivot:
            s += candidats[t][1]  # Contribution au score
            curseurs[t] += 1
            if curseurs[t]<len(postings_list[t])-1:
                candidats[t] = postings_list[t][curseurs[t]]  # Posting suivant pour le terme t
            t+=1
        if s >= theta:  # Réponse possible pour les meilleures k réponses
            heapq.heappush(Ans,(doc_pivot,s))

            if len(Ans) > k:
                Ans.remove(min(Ans, key=lambda x: x[1]))
                theta = Ans[0][1]

    else:  # Impossible d'évaluer cpivot pour le moment
        for t in range(pivot):
            candidats[t], curseurs[t] = seek_to_document(postings_list[t], curseurs[t], doc_pivot)  # Déplacer le pointeur au document cpivot ou suivant

return Ans
```

Algorithme WAND

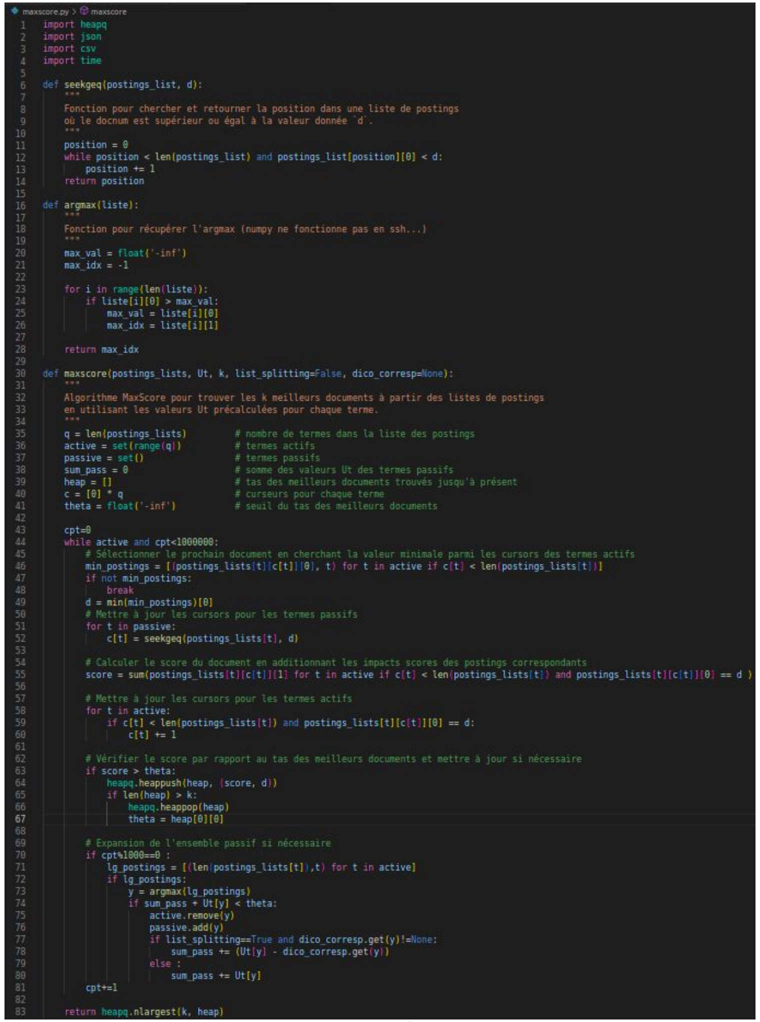


```python
def all_queries(file, postings_clipping=False, list_splitting=False):
    queries = read_queries()
    with open(file, 'r') as f:
        print("load index")
        index = json.load(f)

        for k in [10,1000]:
            print("--- k =",k)
            start_time = time.time()
            cpt=0
            for query in queries:
                cpt+=1
                print(cpt)
                if postings_clipping :
                    WAND(query, index, k, postings_clipping = True)
                else :
                    if list_splitting:
                        WAND(query, index, k, list_splitting = True)
                    else :
                        WAND(query, index, k)
            end_time = time.time()
            total_time = end_time - start_time
            avg_time = total_time / len(queries)
            print("----- Le temps moyen par requête est de : ", avg_time)


def read_queries():
    """
    Permet de lire les 6980 dev queries
    """
    list_queries = []
    with open('queries/msmarco-passage/queries.dev.small.tsv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file, delimiter='\t')
        for row in reader:
            query_text = row[1]
            query_tokens = query_text.split()
            list_queries.append(query_tokens)
    #indices = [5238, 2, 204, 1, 6227]
    #return [list_queries[i] for i in indices]
    return list_queries[1:50]
```



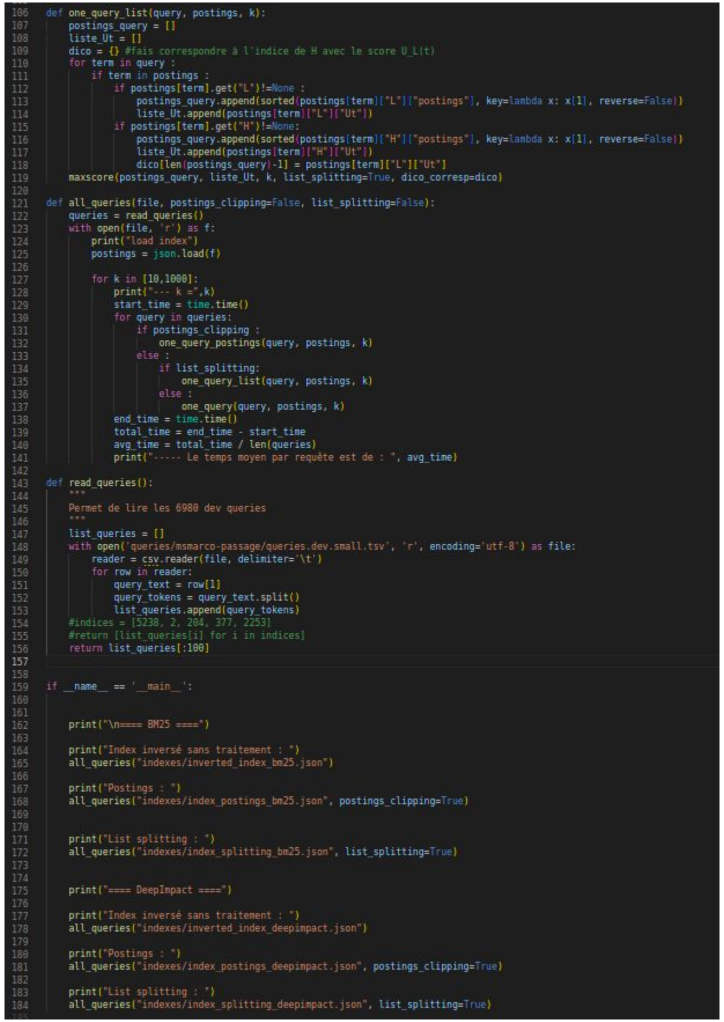
```python
if __name__ == '__main__':
    print("==== DeepImpact ====")

    print("Index inversé sans traitement : ")
    all_queries("indexes/inverted_index_deepimpact.json")

    print("Postings : ")
    all_queries("indexes/index_postings_deepimpact.json", postings_clipping=True)

    print("List splitting : ")
    all_queries("indexes/index_splitting_deepimpact.json", list_splitting=True)


    print("\n==== BM25 ====")

    print("Index inversé sans traitement : ")
    all_queries("indexes/inverted_index_bm25.json")

    print("Postings : ")
    all_queries("indexes/index_postings_bm25.json", postings_clipping=True)


    print("List splitting : ")
    all_queries("indexes/index_splitting_bm25.json", list_splitting=True)
```

Tests WAND



maxscore.py > maxscore

```python
import heapq
import json
import csv
import time

def seekgeq(postings_list, d):
    """
    Fonction pour chercher et retourner la position dans une liste de postings
    où le docnum est supérieur ou égal à la valeur donnée `d`.
    """
    position = 0
    while position < len(postings_list) and postings_list[position][0] < d:
        position += 1
    return position

def argmax(liste):
    """
    Fonction pour récupérer l'argmax (numpy ne fonctionne pas en ssh...)
    """
    max_val = float('-inf')
    max_idx = -1

    for i in range(len(liste)):
        if liste[i][0] > max_val:
            max_val = liste[i][0]
            max_idx = liste[i][1]

    return max_idx

def maxscore(postings_lists, Ut, k, list_splitting=False, dico_corresp=None):
    """
    Algorithme MaxScore pour trouver les k meilleurs documents à partir des listes de postings
    en utilisant les valeurs Ut précalculées pour chaque terme.
    """
    q = len(postings_lists)         # nombre de termes dans la liste des postings
    active = set(range(q))          # termes actifs
    passive = set()                 # termes passifs
    sum_pass = 0                    # somme des valeurs Ut des termes passifs
    heap = []                       # tas des meilleurs documents trouvés jusqu'à présent
    c = [0] * q                     # curseurs pour chaque terme
    theta = float('-inf')           # seuil du tas des meilleurs documents

    cpt=0
    while active and cpt<1000000:
        # Sélectionner le prochain document en cherchant la valeur minimale parmi les cursors des termes actifs
        min_postings = [(postings_lists[t][c[t]][0], t) for t in active if c[t] < len(postings_lists[t])]
        if not min_postings:
            break
        d = min(min_postings)[0]
        # Mettre à jour les cursors pour les termes passifs
        for t in passive:
            c[t] = seekgeq(postings_lists[t], d)

        # Calculer le score du document en additionnant les impacts scores des postings correspondants
        score = sum(postings_lists[t][c[t]][1] for t in active if c[t] < len(postings_lists[t]) and postings_lists[t][c[t]][0] == d )

        # Mettre à jour les cursors pour les termes actifs
        for t in active:
            if c[t] < len(postings_lists[t]) and postings_lists[t][c[t]][0] == d:
                c[t] += 1

        # Vérifier le score par rapport au tas des meilleurs documents et mettre à jour si nécessaire
        if score > theta:
            heapq.heappush(heap, (score, d))
            if len(heap) > k:
                heapq.heappop(heap)
                theta = heap[0][0]

        # Expansion de l'ensemble passif si nécessaire
        if cpt%1000==0 :
            lg_postings = [(len(postings_lists[t]),t) for t in active]
            if lg_postings:
                y = argmax(lg_postings)
                if sum_pass + Ut[y] < theta:
                    active.remove(y)
                    passive.add(y)
                    if list_splitting==True and dico_corresp.get(y)!=None:
                        sum_pass += (Ut[y] - dico_corresp.get(y))
                    else :
                        sum_pass += Ut[y]
        cpt+=1

    return heapq.nlargest(k, heap)
```



```python
def one_query_list(query, postings, k):
    postings_query = []
    liste_Ut = []
    dico = {} #fais correspondre à l'indice de H avec le score U_L(t)
    for term in query :
        if term in postings :
            if postings[term].get("L")!=None :
                postings_query.append(sorted(postings[term]["L"]["postings"], key=lambda x: x[1], reverse=False))
                liste_Ut.append(postings[term]["L"]["Ut"])
            if postings[term].get("H")!=None:
                postings_query.append(sorted(postings[term]["H"]["postings"], key=lambda x: x[1], reverse=False))
                liste_Ut.append(postings[term]["H"]["Ut"])
                dico[len(postings_query)-1] = postings[term]["L"]["Ut"]
    maxscore(postings_query, liste_Ut, k, list_splitting=True, dico_corresp=dico)

def all_queries(file, postings_clipping=False, list_splitting=False):
    queries = read_queries()
    with open(file, 'r') as f:
        print("load index")
        postings = json.load(f)

        for k in [10,1000]:
            print("--- k =",k)
            start_time = time.time()
            for query in queries:
                if postings_clipping :
                    one_query_postings(query, postings, k)
                else :
                    if list_splitting:
                        one_query_list(query, postings, k)
                    else :
                        one_query(query, postings, k)
            end_time = time.time()
            total_time = end_time - start_time
            avg_time = total_time / len(queries)
            print("----- Le temps moyen par requête est de : ", avg_time)

def read_queries():
    """
    Permet de lire les 6980 dev queries
    """
    list_queries = []
    with open('queries/msmarco-passage/queries.dev.small.tsv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file, delimiter='\t')
        for row in reader:
            query_text = row[1]
            query_tokens = query_text.split()
            list_queries.append(query_tokens)
    #indices = [5238, 2, 204, 377, 2253]
    #return [list_queries[i] for i in indices]
    return list_queries[:100]


if __name__ == '__main__':


    print("\n==== BM25 ====")

    print("Index inversé sans traitement : ")
    all_queries("indexes/inverted_index_bm25.json")

    print("Postings : ")
    all_queries("indexes/index_postings_bm25.json", postings_clipping=True)


    print("List splitting : ")
    all_queries("indexes/index_splitting_bm25.json", list_splitting=True)


    print("==== DeepImpact ====")

    print("Index inversé sans traitement : ")
    all_queries("indexes/inverted_index_deepimpact.json")

    print("Postings : ")
    all_queries("indexes/index_postings_deepimpact.json", postings_clipping=True)

    print("List splitting : ")
    all_queries("indexes/index_splitting_deepimpact.json", list_splitting=True)
```

MaxScore