Université de Strasbourg

Recherche d'information

Recherche dense et hybride

Enzo Doyen

2025 - LGC6KM43 - M2

Méthodes de récupération de documents

Plusieurs méthodes peuvent être utilisées pour récupérer des documents pertinents à partir d'une requête formulée en langage naturel :

- **recherche par mots-clés** : correspondance exacte (ou quasi exacte) entre les termes de la requête et ceux présents dans les documents. La pertinence peut être estimée grâce à des **modèles épars** comme TF-IDF ou BM25, qui pondèrent l'importance des mots en fonction de leur fréquence dans le document et dans la collection.
- **recherche sémantique/dense** : elle vise à interpréter le sens de la requête afin d'identifier des documents pertinents, même si les termes employés diffèrent. Cette approche s'appuie sur des **modèles denses** qui exploitent des plongements vectoriels (au niveau des mots ou de la phrase) pour mesurer la similarité sémantique entre la requête et les documents.

Méthodes de récupération de documents

Plusieurs méthodes peuvent être utilisées pour récupérer des documents pertinents à partir d'une requête formulée en langage naturel :

- **recherche par mots-clés** : correspondance exacte (ou quasi exacte) entre les termes de la requête et ceux présents dans les documents. La pertinence peut être estimée grâce à des **modèles épars** comme TF-IDF ou BM25, qui pondèrent l'importance des mots en fonction de leur fréquence dans le document et dans la collection.
- **recherche sémantique/dense** : elle vise à interpréter le sens de la requête afin d'identifier des documents pertinents, même si les termes employés diffèrent. Cette approche s'appuie sur des **modèles denses** qui exploitent des plongements vectoriels (au niveau des mots ou de la phrase) pour mesurer la similarité sémantique entre la requête et les documents.

🕮 Plan

I. Modèles de recherche dense

II. Modèles de recherche hybride

I. Modèles de recherche dense

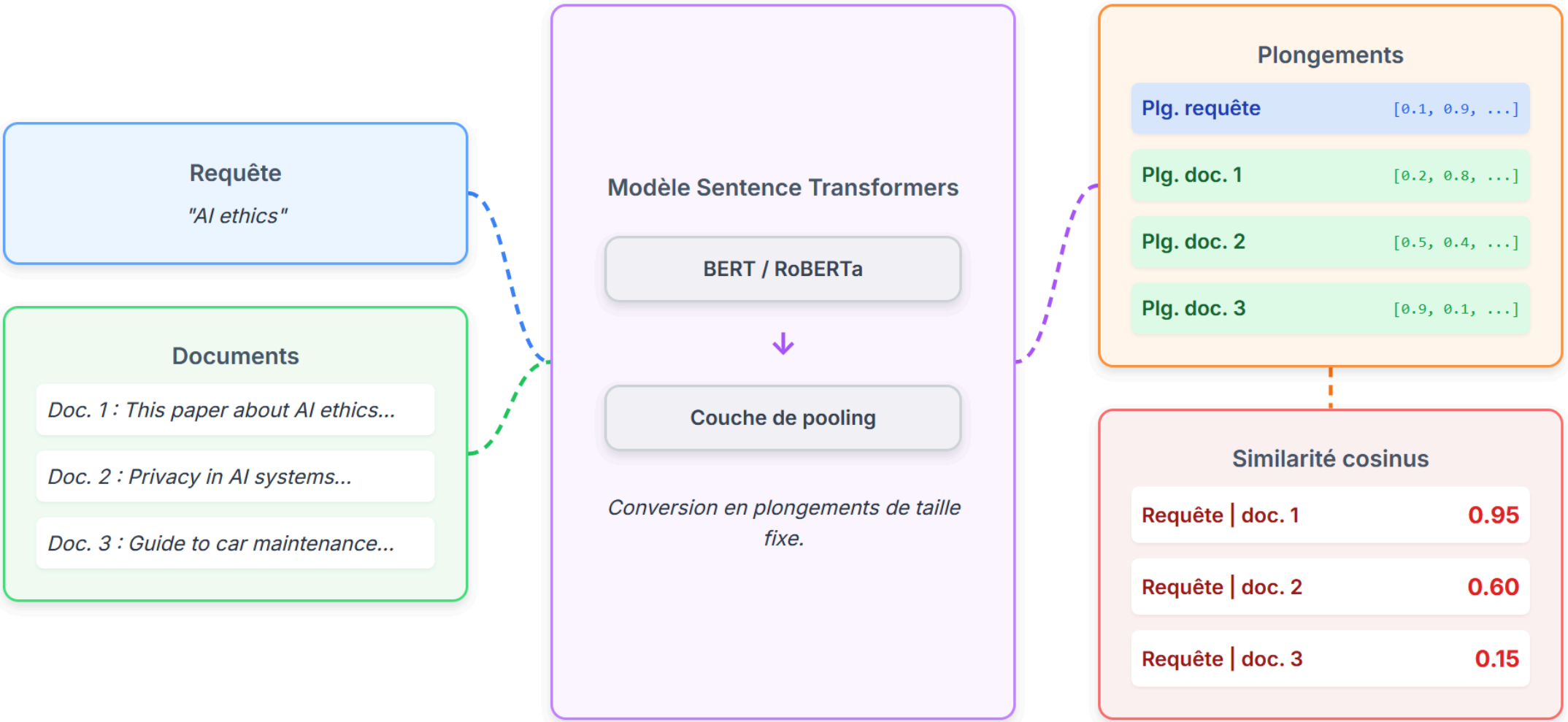
Recherche dense

Les modèles que nous avons vus jusqu'à présent (TF-IDF et BM25, dits « **modèles épars** ») ne permettent pas de capturer l'aspect sémantique des documents et des requêtes. Cela signifie que le sens des mots est ignoré, et que les synonymes ne sont pas pris en compte pour la recherche.

Les **modèles denses** permettent de surmonter ces limitations en capturant les représentations vectorielles des mots et des documents.

Recherche dense

Des bibliothèques Python telles que `sentence-transformers` [?] permettent de créer facilement ces représentations vectorielles.


Requête
"AI ethics"
Documents
Doc. 1 : This paper about AI ethics...
Doc. 2 : Privacy in AI systems...
Doc. 3 : Guide to car maintenance...
Modèle Sentence Transformers
BERT / RoBERTa
Couche de pooling
Conversion en plongements de taille fixe.
Plongements
Plg. requête [0.1, 0.9, ...]
Plg. doc. 1 [0.2, 0.8, ...]
Plg. doc. 2 [0.5, 0.4, ...]
Plg. doc. 3 [0.9, 0.1, ...]
Similarité cosinus
Requête | doc. 1 0.95
Requête | doc. 2 0.60
Requête | doc. 3 0.15

Recherche dense

Des bibliothèques Python telles que `sentence-transformers` [?] permettent de créer facilement ces représentations vectorielles :

Python

```python
model = SentenceTransformer(MODEL)
doc_embeddings = model.encode_document(documents, convert_to_tensor=True)
query_embedding = model.encode_query(query, convert_to_tensor=True)
similarity_scores = model.similarity(query_embedding, doc_embeddings)[0]
scores, indices = torch.topk(similarity_scores, k=TOP_K)
```

Recherche dense : recherche symétrique/ asymétrique

On dit qu'une recherche est **symétrique** quand la requête et les documents ont à peu près la même longueur.

On dit qu'une recherche est **asymétrique** quand la requête est bien plus courte que les documents (p. ex., question ou recherche par mots-clés).

Recherche dense : recherche symétrique/ asymétrique

En fonction du type de recherche, les fonctions de sentence-transformers à utiliser diffèrent :

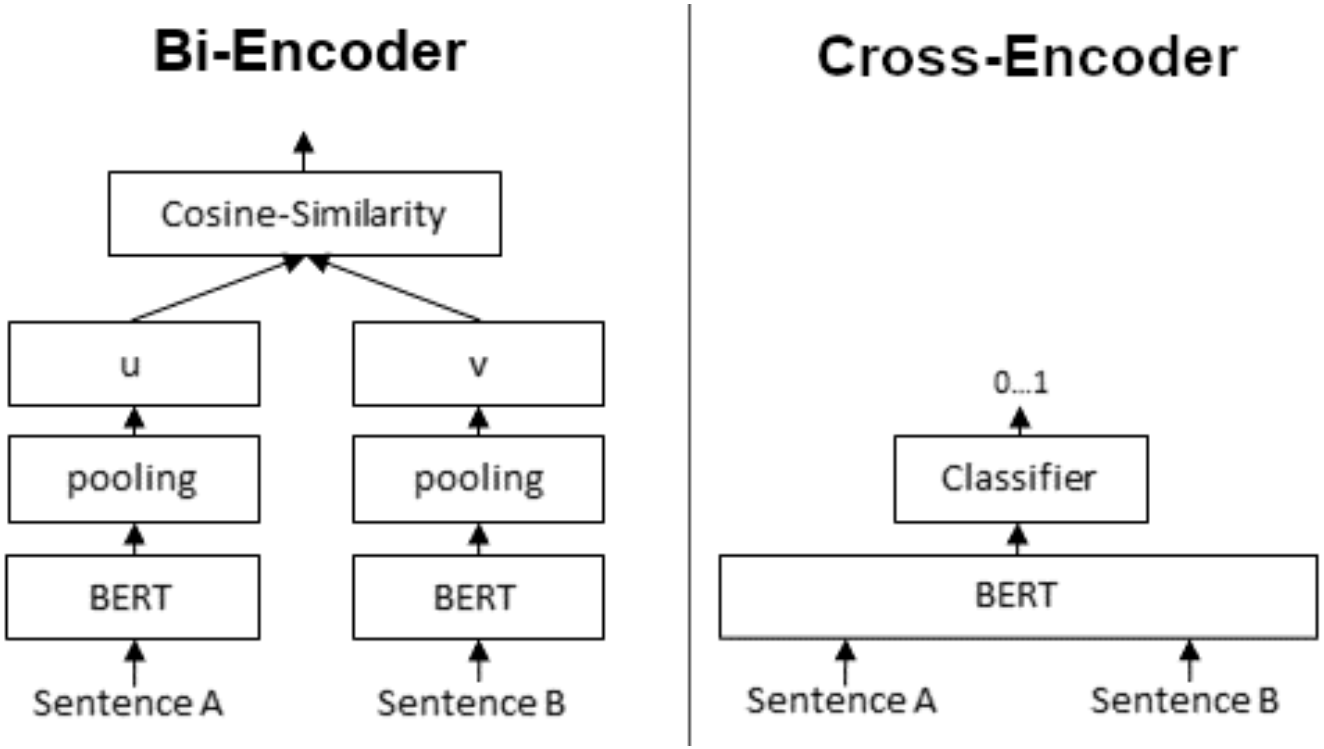
- en cas de recherche symétrique, on utilise `model.encode()` ;
- en cas de recherche asymétrique, on utilise `model.encode_query()` pour la requête et `model.encode_document()` pour les documents.

Plus d'informations : **https://sbert.net/examples/sentence_transformer/applications/semantic-search/README.html#symmetric-vs-asymmetric-semantic-search**

Recherche dense : types d'encodeurs/modèles

- **bi-encodeur** : un modèle qui encode la requête et les documents indépendamment, avec des plongements distincts. La similarité entre la requête et les documents est mesurée selon la distance des plongements.
- **cross-encodeur** : un modèle qui encode la requête et les documents ensemble et qui donne un score de similarité.

Recherche dense : types d'encodeurs/modèles



Source : https://sbert.net/examples/cross_encoder/applications/README.html

Recherche dense : types d'encodeurs/modèles

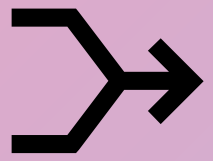
Bi-encodeurs

- https://huggingface.co/google/embeddinggemma-300m (multilingue)
- https://huggingface.co/intfloat/multilingual-e5-large (multilingue ; **(Wang et al., 2024)**)
- https://huggingface.co/antoinelouis/biencoder-mMiniLMv2-L6-mmarcoFR (français)

Cross-encodeurs

En français (basés sur le modèle CamemBERT **(Martin et al., 2020)**) :

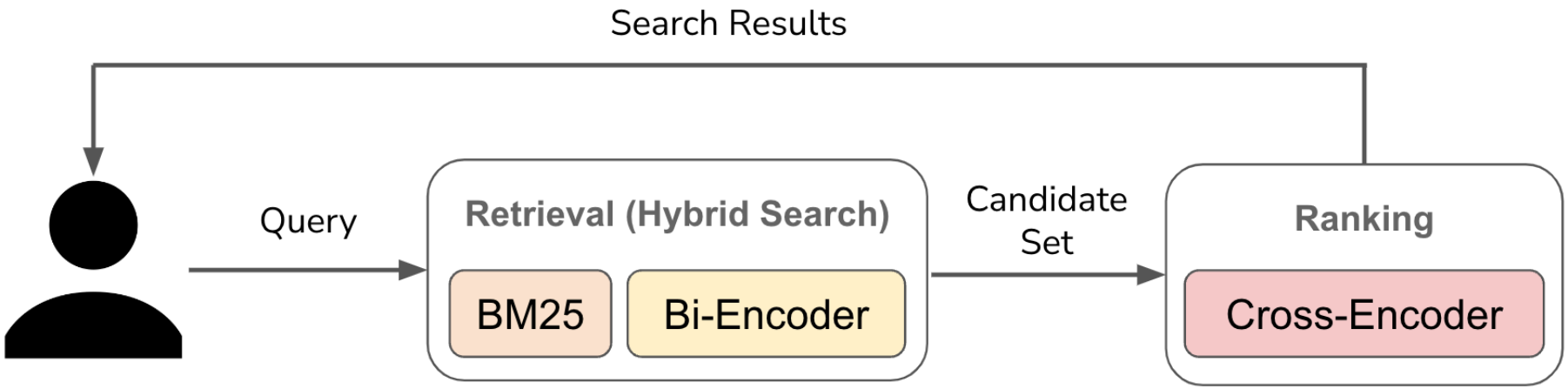
- https://huggingface.co/antoinelouis/crossencoder-camembert-base-mmarcoFR
- https://huggingface.co/dangvantuan/CrossEncoder-camembert-large



II. Modèles de recherche hybride

Recherche hybride

La recherche hybride combine les scores des modèles épars (comme BM25) et des modèles denses (comme ceux basés sur des plongements).



Source : https://cameronrwolfe.substack.com/p/the-basics-of-ai-powered-vector-search

Recherche hybride

La recherche hybride combine les scores des modèles épars (comme BM25) et des modèles denses (comme ceux basés sur des plongements).

On ajoute une étape supplémentaire, celle du **reclassement** (*reranking*), qui sert à obtenir les documents les plus pertinents d'après les meilleurs résultats des deux modèles.

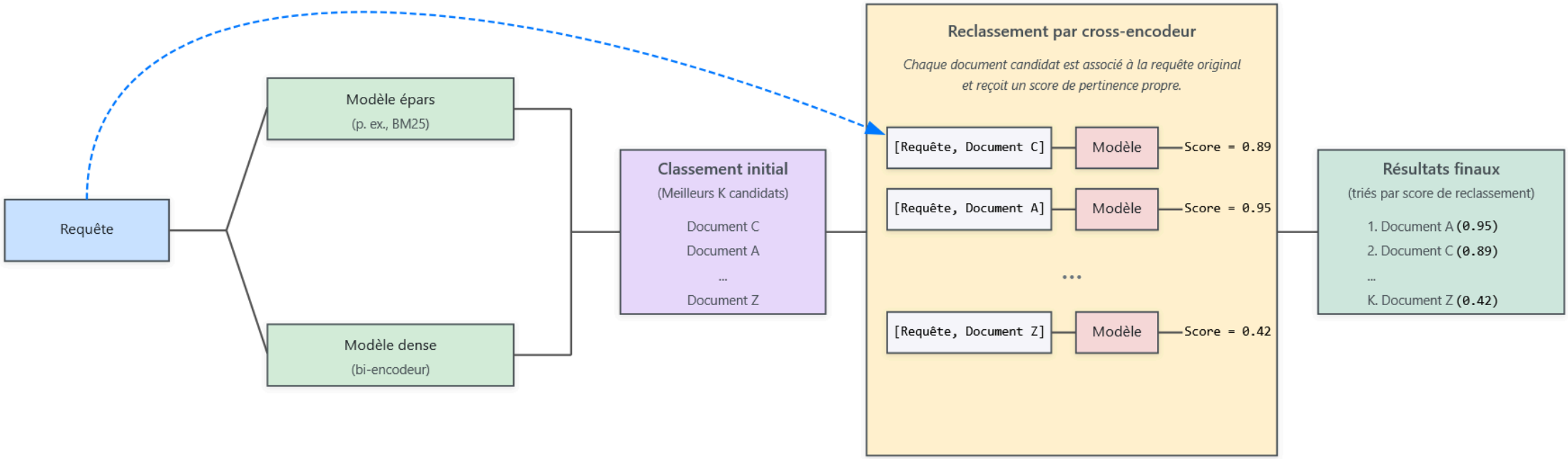
Recherche hybride : reclassement

L'étape de reclassement peut se faire de plusieurs manières :

- **reclassement par cross-encodeur** : on utilise un modèle de type cross-encodeur pour calculer un score de pertinence entre la requête et les documents récupérés par les modèles épars et denses.
- **reclassement par RRF (Ranked Retrieval Fusion)** : algorithme qui unifie les classements des différents modèles.
- **reclassement par modèle de langue à base d'instructions** (p. ex., GPT-5) : méthode explorée plus récemment **(Déjean et al., 2024 ; Sun et al., 2024)**. [±]

Ces méthodes ne sont pas exclusives et peuvent être combinées (RRF pour unifier, puis cross-encodeur pour obtenir un classement plus précis).

Recherche hybride : reclassement par cross-encodeur



Recherche hybride : reclassement par RRF

L'algorithme RRF (Ranked Retrieval Fusion) est défini comme suit :

$$\mathrm{RRF}(d) = \sum_{s \in S} \frac{1}{k + \mathrm{rank}_s(d)}$$

Où d est un document, S un ensemble de systèmes de recherche, $\mathrm{rank}_s(d)$ le rang du document d dans la liste des résultats du système s, et k une constante utilisée pour atténuer l'influence des rangs élevés (généralement 60).

Recherche hybride : reclassement par RRF

Épars	Dense	Score RRF (k=60)	Final
1	2	$\frac{1}{k+1} + \frac{1}{k+2} = \frac{1}{61} + \frac{1}{62} \approx 0.03252$	1
2	1	$\frac{1}{k+2} + \frac{1}{k+1} = \frac{1}{62} + \frac{1}{61} \approx 0.03252$	1
3	4	$\frac{1}{k+3} + \frac{1}{k+4} = \frac{1}{63} + \frac{1}{64} \approx 0.03150$	4
5	2	$\frac{1}{k+5} + \frac{1}{k+2} = \frac{1}{65} + \frac{1}{62} \approx 0.03151$	3

Recherche hybride : reclassement par RRF, implémentation en Python

Soit X une liste de tuples `(ID document, score)` d'un système donné, préalablement triée par score décroissant :

```python
rrf_scores = defaultdict(float)
k = 60
for rank, (doc_id, _) in enumerate(X, start=1):
    rrf_scores[doc_id] += 1 / (k + rank)
```

Il suffit de répéter la même chose pour chaque système puis de trier le dictionnaire pour obtenir les meilleurs scores.

⊕ Ressources complémentaires

- *Sentence Embeddings. Cross-encoders and Re-ranking* : **https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings2/**
- *Awesome Information Retrieval* : **https://github.com/harpribot/awesome-information-retrieval**

Bibliographie

Déjean, H., Clinchant, S., et Formal, T. (mars 2024). *A Thorough Comparison of Cross-Encoders and LLMs for Reranking SPLADE* (Numéro arXiv:2403.10407). arXiv. 10.48550/arXiv.2403.10407

Manning, C. D., Raghavan, P., et Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Martin, L., Muller, B., Suárez, P. J. O., Dupont, Y., Romary, L., Clergerie, É. V. de la, Seddah, D., et Sagot, B. (2020). CamemBERT: A Tasty French Language Model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203–7219. 10.18653/v1/2020.acl-main.645

Mitra, B. (2018). *An Introduction to Neural Information Retrieval* (Numéro v.41). Now Publishers.

Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D., et Ren, Z. (décembre 2024). *Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents* (Numéro arXiv:2304.09542). arXiv. 10.48550/arXiv.2304.09542

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., et Wei, F. (février 2024). *Multilingual E5 Text Embeddings: A Technical Report* (Numéro arXiv:2402.05672). arXiv. 10.48550/arXiv.2402.05672

Zhai, C., et Massung, S. (juin 2016). *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Association for Computing Machinery and Morgan & Claypool. 10.1145/2915031