# LearnInBits

## Semantic Video Search: Methods and Evaluation

**CentraleSupélec**
Axel Voyer


Dominante INFONUM 2023-2024

# Table des matières

# 1 Analyse des choix techniques

Après avoir brièvement présenté les fonctionnalités implémentées et la structure générale de l'application, nous allons passer en revue les choix techniques pour chaque aspect de la solution.

## 1.1 Speech-to-Text

La première étape de la création d'une base de données indexée est le traitement des vidéos. Comme expliqué précédemment, nous avons simplifié le traitement des vidéos en se limitant à la transcription de l'audio en texte. Pour cette tâche, plusieurs options d'implémentation sont possibles. La première consiste à utiliser l'API de transcription gratuite de Google : Google Speech Recognition. Celle-ci présente cependant des limitations, notamment une qualité moyenne de la sortie, une durée maximale de traitement de 5 minutes par audio, et le fait qu'elle ne permet pas la récupération des timestamps ni de la ponctuation. Nous verrons donc comment, grâce à des étapes de traitement effectuées en amont et en aval de l'utilisation de l'API, nous avons pu implémenter une solution fonctionnelle. La deuxième option d'implémentation est de choisir parmi les API de transcription payantes disponibles sur le marché. Après une comparaison des performances, nous avons décidé d'explorer la solution Whisper d'OpenAI, qui permet de traiter de longs audios, d'utiliser un prompt pour ajouter un contexte, et qui retourne la ponctuation et des timestamps. Nous avons implémenté les deux options et les avons testées sur des audios issus de notre base de données de test, en comparant les textes obtenus à une transcription manuellement corrigée. Nous examinerons finalement la comparaison des résultats entre les deux implémentations.

### 1.1.1 Transcription par Google Speech Recognition

**Préparation de l'audio** : Initialement, la vidéo est convertie en un fichier audio avec un taux d'échantillonnage précis pour assurer la compatibilité avec l'API Google Speech Recognition. L'étape suivante consiste à découper l'audio en segments de moins de 5 minutes pour éviter les coupures en milieu de phrase, afin de conserver un contexte suffisant pour chaque audio. Cette segmentation se fait sur les silences, générant une liste d'audios traitables indépendamment et facilitant le calcul des timestamps pour chaque segment.

**Optimisation des paramètres de segmentation** : Avant le découpage complet sur les silences, une phase de test sur un extrait de cinq minutes permet d'ajuster les paramètres idéaux pour la segmentation de l'audio. L'objectif est d'atteindre un équilibre parfait entre la longueur des segments, en modulant la durée des silences et la sensibilité de leur détection. Si les paramètres initiaux ne sont pas satisfaisants, ils sont ajustés et les tests sont réitérés jusqu'à l'obtention des résultats désirés.

**Transcription des segments** : Après ajustement des paramètres, l'audio est

divisé en segments qui sont transcrits via Google Speech Recognition. Chaque segment produit un texte auquel on attribue des timestamps de début et de fin, basés sur le découpage préalable de l'audio.

**Correction de la ponctuation et des majuscules** : La transcription brute est ensuite améliorée par l'ajout de la ponctuation et des majuscules, ce qui augmente la lisibilité et la cohérence du texte. Cette étape est cruciale pour garantir une bonne interprétation du texte par les algorithmes de recherche ultérieurs, notamment ceux basés sur les embeddings. Le modèle utilisé pour la recréation de la ponctuation est un modèle multilingue basé sur des Transformers, spécialement développé pour restaurer la ponctuation dans les transcriptions de langage parlé. Il a été entraîné sur le dataset Europarl de la SEPP-NLG Shared Task, consitué de discours politiques, et peut prédire la ponctuation pour les textes en anglais, italien, français et allemand [Guhr et al., 2021].

**Ajustement des timestamps** : Pour les traitements ultérieurs, il est nécessaire de disposer des timestamps pour les début et fin de chaque phrase. Les timestamps initialement obtenus à partir des silences de l'audio sont donc réajustés pour correspondre aux phrases dans le texte transcrit, en utilisant une méthode d'approximation linéaire basée sur le rapport entre le nombre de mots et la durée des segments audio.

**Enregistrement du résultat** : Finalement, le texte transcrit et les timestamps associés à chaque phrase sont sauvegardés, prêts à être découpés en chunks sémantiques pour des traitements ultérieurs.

### 1.1.2 Transcription par Whisper d'OpenAI

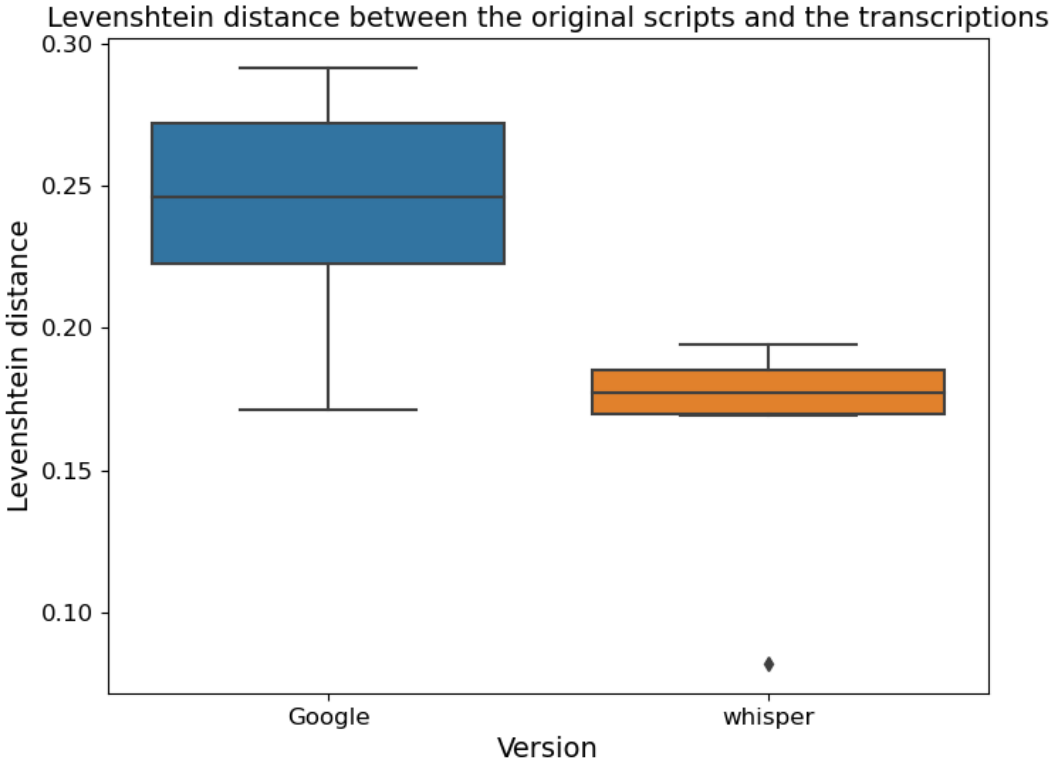
Klee étant déjà familier avec les produits d'OpenAI, nous avons décidé d'utiliser l'outil de transcription Whisper. Le prétraitement de l'audio est ici plus léger étant donné que l'API accepte des audios bien plus longs, et que la sortie est fournie avec les timestamps pour chaque phrase. Il est également possible de donner un prompt en entrée permettant d'indiquer le contexte. Cela permet notamment la bonne compréhension de certains mots spécifiques qui sont difficilement reconnus par l'API Google Speech Recognition, tels que "l'IA" ou "le Cloud", souvent confondus avec d'autres mots français. Cependant, l'utilisation de l'API d'OpenAI implique un coût financier, s'élevant à 0,36$ par heure d'audio à transcrire. L'étape de la transcription n'étant effectuée qu'une fois lors de la construction de la base de données, ce prix peut être considéré comme raisonnable.

Whisper a été entraîné sur 680 000 heures d'audio, couvrant l'anglais ainsi que 98 autres langues, lui permettant de transcrire avec précision une vaste gamme de termes spécifiques dans différents domaines. Le modèle est multilingue et s'adapte automatiquement à la langue de l'audio sans nécessiter de préciser la langue attendue en paramètre, et est efficace dans des contextes difficiles, notamment bruyants ou avec un fort accent. [Radford et al., 2022]

### 1.1.3 Comparaison des deux implémentations de transcription

Afin de comparer la qualité de la transcription, nous avons évalué chaque modèle sur six audios tests. Ces audios sont issus de vidéos de formations fournies par Klee, retranscrites à la main par nos soins, pour un total de 19 minutes soit 3 514 mots. La distance de Levenshtein sur les mots est ensuite utilisée pour comparer les deux modèles.

On observe qu'en plus de fournir des timestamps plus précis, l'utilisation de l'API d'OpenAI permet d'obtenir de meilleurs résultats sur la qualité de la transcription finale.



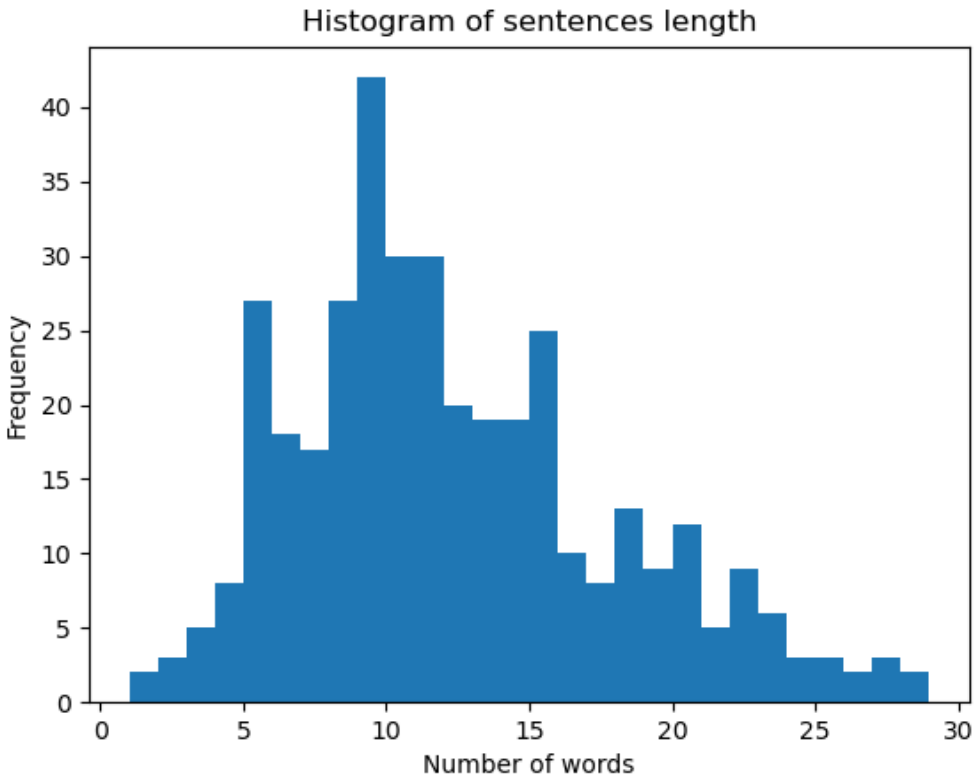

Par la suite, nous n'utiliserons que les transcriptions obtenues à partir de l'API d'OpenAI et le model correspondant Whisper.

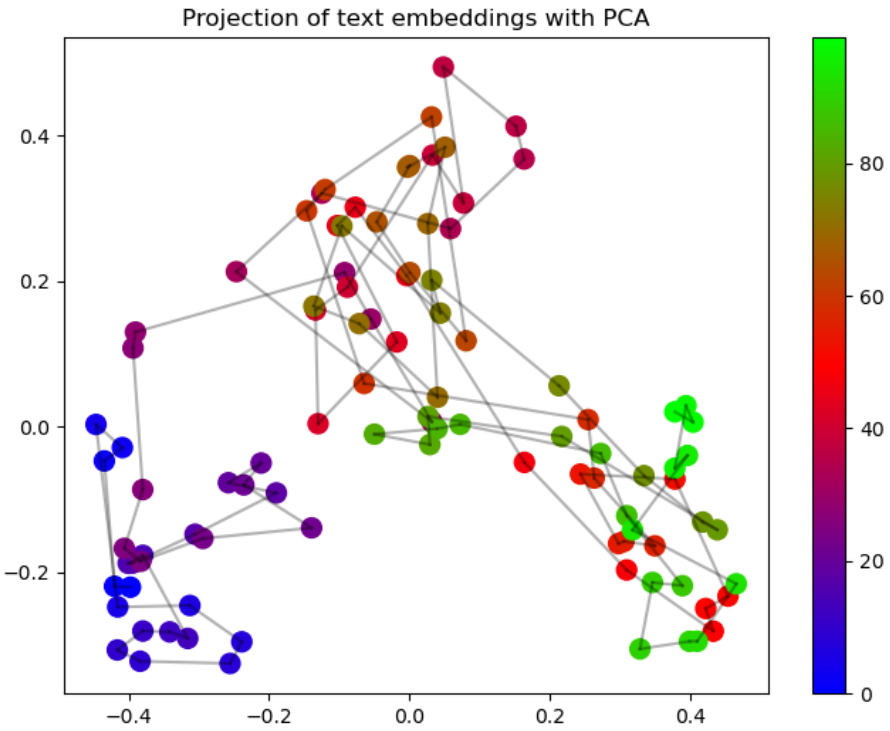
## 1.2 Découpage par sémantique des transcriptions en chunks

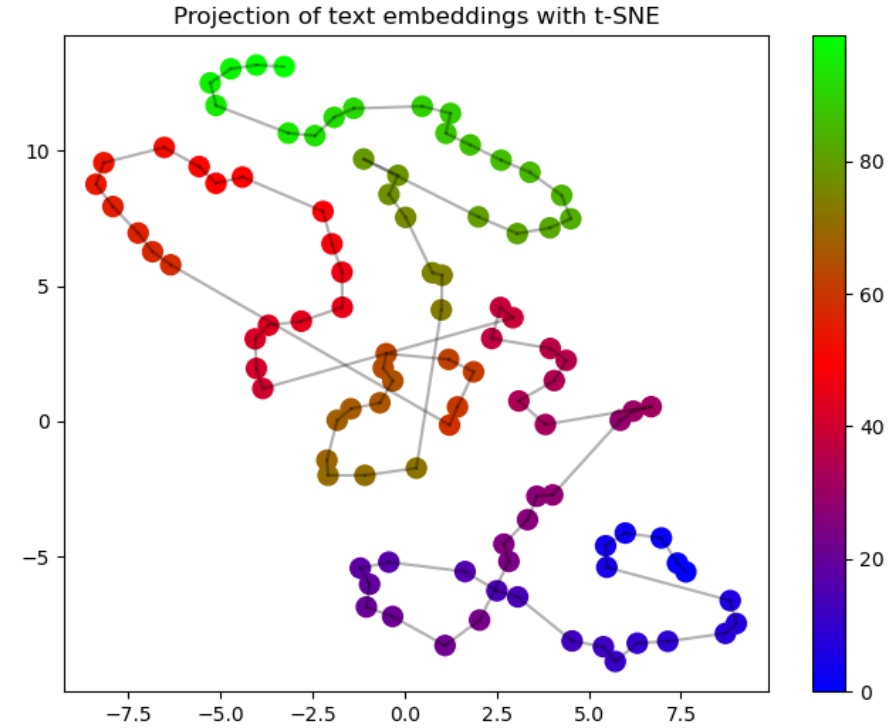
L'étape qui suit consiste à découper les transcriptions en chunks de telle sorte que chaque chunk représente un sujet distinct dans la formation, et que les longueurs des chunks correspondent à des tailles dans un fourchette prédéfinies. Après plusieurs itérations et améliorations, nous présentons la méthode finale retenue ci-dessous.

**Embedding des phrases** : La première étape consiste à transformer chaque phrase obtenue lors de l'étape de transcription en un vecteur dans un espace latent, ou embedding. Les phrases, étant potentiellement courtes (cf. l'histogramme du

nombre de mots par phrase lors du traitement d'une vidéo test ci-dessous), elles sont enrichies en ajoutant la phrase précédente et la suivante pour fournir du contexte. En utilisant le modèle Sentence-BERT [Reimers and Gurevych, 2019] (plus d'informations sur ce modèle dans la partie 5.3.4) ou l'API d'OpenAI, chaque phrase enrichie est transformée en un embedding représentant sa sémantique. L'hypothèse est que les phrases se "suivent" dans l'espace latent, jusqu'à ce qu'un "saut" entre deux phrases consécutives marque un changement de sujet dans le discours. Afin de vérifier si cette hypothèse est satisfaisante, nous visualisons une projection en 2D des embeddings des 100 premières phrases, en utilisant d'abord un PCA, puis la méthode t-SNE. Les couleurs permettent de visualiser l'avancement dans le discours, montrant que les phrases consécutives ont tendance à suivre un fil directeur, jusqu'à ce qu'un "saut" indique un changement de sujet.

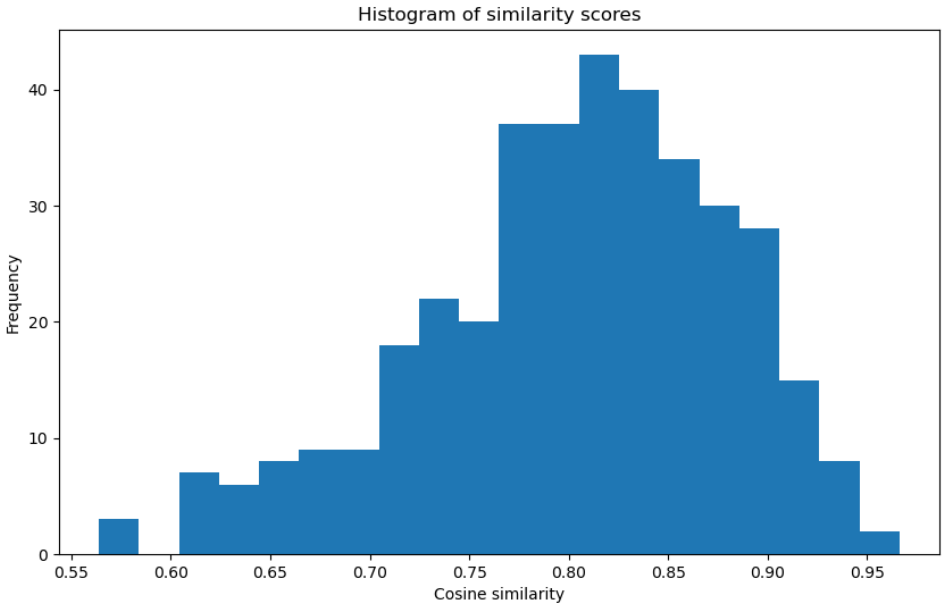




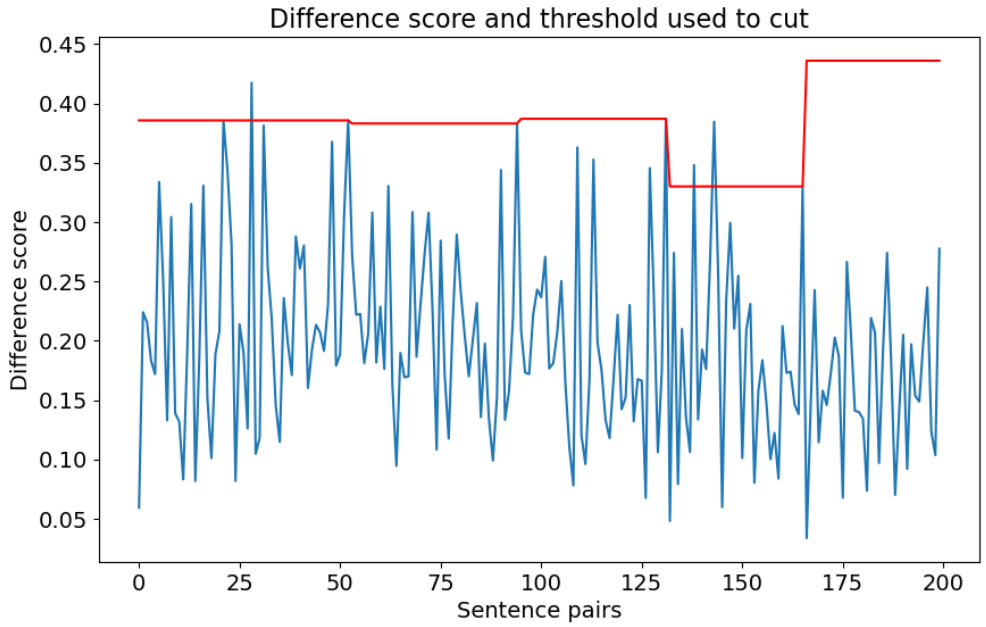



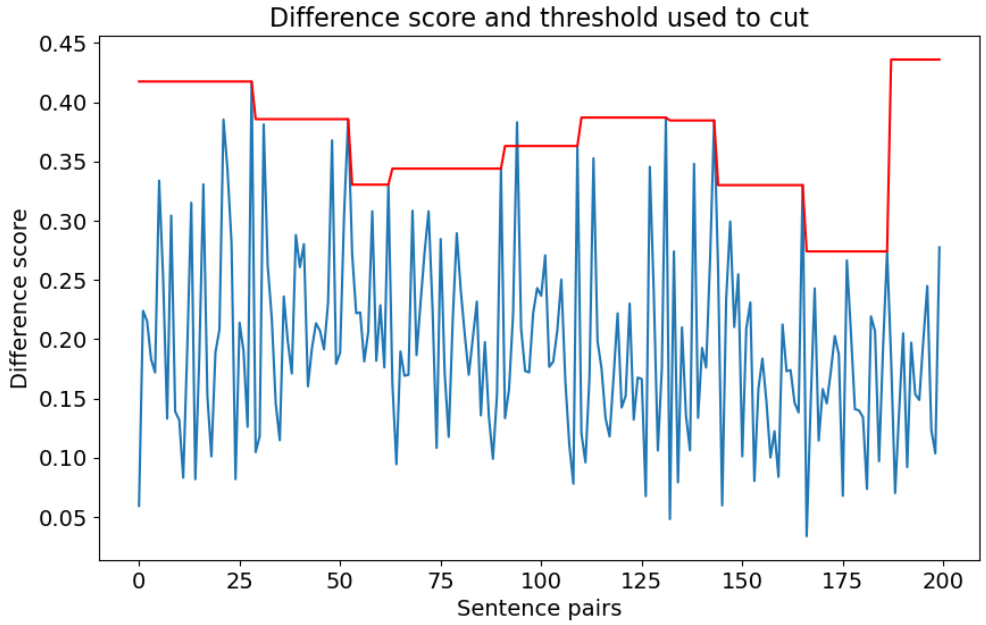

**Similarité et distance entre deux phrases consécutives** : La deuxième étape consiste à calculer la similarité cosinus entre chaque pairs de phrases consécutives. On peut visualiser ci-dessous l'histogramme de la similarité entre chaque pairs

pour une vidéo de test. On observe qu'une grande majorité des phrases consécutives sont proches les unes des autres, et seules quelques unes représentent une rupture plus importante (<0,70 de similarité cosinus par exemple). Ce sont à ces ruptures que nous souhaitons découper le texte.

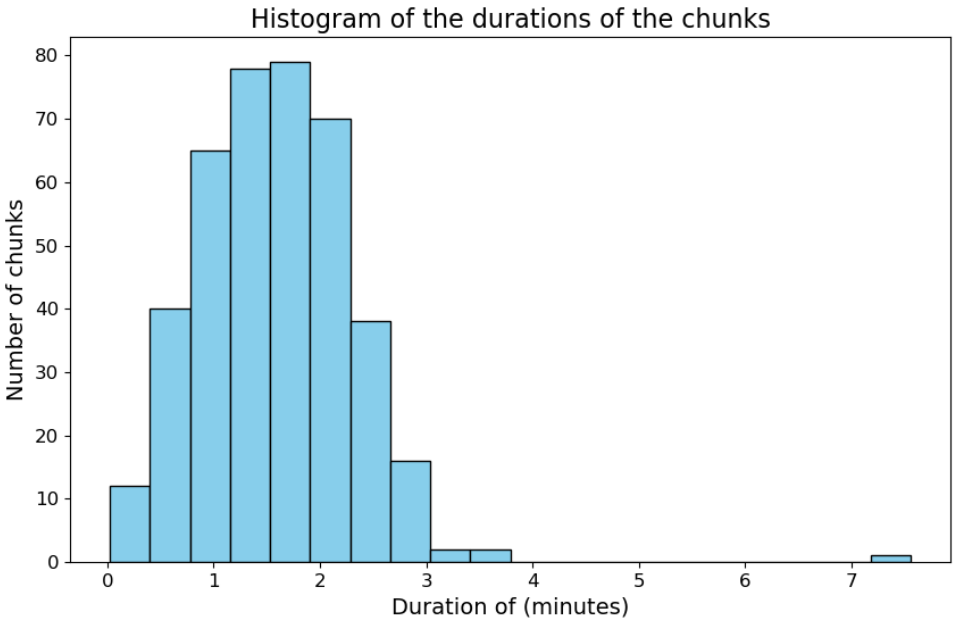



**Découpage en chunks** : Pour sélectionner les positions des séparations, nous commençons par définir une longueur minimale et maximale des chunks en termes de nombre de mots. Cela permet de contraindre la taille que feront les extraits de vidéo, et d'assurer une longueur suffisamment grande pour ne pas manquer de contexte lorsque seul l'extrait est visionné, mais suffisamment courte pour ne présenter à l'utilisateur que l'extrait répondant réellement à sa requête. Pour chaque séparation, la position de coupure est sélectionnée en prenant la plus grande différence entre les phrases consécutives au sein de la fenêtre de tailles acceptables après la coupure précédente. Ci-dessous, nous visualisons la distance entre les phrases consécutives (1-similarité) en bleu, et en rouge la limite utilisée pour le découpage du chunk, pour deux paramètres de longueur de chunk différents. On observe que le découpage n'a parfois pas lieu malgré deux phrases consécutives très différentes, lorsque le chunk qui aurait été obtenu aurait été trop court. Par ailleurs, on observe également que le seuil de différence utilisé pour définir la position de la coupure varie entre les chunks, étant donné que certaines parties du discours portent sur des sujets moins variés que d'autres.




**Calcul des timestamps pour les chunks générés** : À partir des timestamps de chaque phrase obtenue lors de la transcription, nous calculons les timestamps des chunks en regroupant les phrases qui font partie du même chunk. Nous obtenons ainsi les temps de début et de fin pour chaque chunk. Voici ci-dessous un histogramme des longueurs des extraits obtenus sur 10 heures de vidéo à partir de cette méthode. On observe une répartition en cloche, avec un extrait qui sort du lot, dû à un silence dans l'audio d'origine.




## 1.3 Algorithmes de recherche d'information textuelle

Une fois les vidéos transcrites et les transcriptions découpées en chunks, on se ramène à un problème de recherche d'information dans une base de documents textuels. Ce problème est très classique en informatique et de nombreuses méthodes ont été développées au cours du temps. Nous en avons implémenté plusieurs afin de comparer leur efficacité dans notre situation. Nous verrons ainsi les méthodes vectorielles basées sur la fréquence des mots avec le TF-IDF, puis une méthode probabiliste : le BM25. Nous verrons ensuite l'utilisation des n-grams pour améliorer la méthode basée sur les fréquences de mots. Enfin, nous aborderons des méthodes plus récentes basées sur le deep-learning et notamment l'embedding des documents. Nous verrons également le cross-encoder qui permet une comparaison plus fine de deux documents. Enfin, nous utiliserons l'API d'OpenAI pour transformer les paragraphes en embeddings et comparer le résultat avec les autres méthodes gratuites précédentes.

### 1.3.1 Méthode vectorielle simple : TF-IDF

Le TF-IDF est une mesure statistique qui vise à évaluer l'importance d'un mot dans un document par rapport à une collection de documents.

La fréquence des termes (TF) mesure la fréquence d'un mot dans un document. Cela permet de connaître les termes les plus significatifs d'un document. La formule pour calculer le TF d'un mot $t$ dans un document $d$ est la suivante :

$$TF(t,d) = \frac{\text{Nombre de fois que le terme } t \text{ apparaît dans le document } d}{\text{Nombre total de termes dans le document } d}$$

L'inverse de la fréquence documentaire (IDF) vise à mesurer l'importance d'un terme dans l'ensemble du corpus. L'idée est que si un terme apparaît dans beaucoup de documents, il n'est pas un bon discriminant et est donc moins important. La formule de l'IDF est :

$$IDF(t,D) = \log\left(\frac{\text{Nombre total de documents}}{\text{Nombre de documents contenant le terme } t}\right)$$

La mesure TF-IDF d'un terme est simplement le produit de ses mesures TF et IDF. Elle vise à réduire l'impact des termes fréquents tout en prenant en compte l'importance des termes spécifiques au document. La formule du TF-IDF pour un terme $t$ dans un document $d$ par rapport à un corpus $D$ est :

$$TF\text{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

L'indexation de la base de données consiste alors à calculer, pour chaque document, le vecteur correspondant, pour lequel chaque coefficient est le score TF-IDF du mot $i$. La recherche s'effectue alors en transformant la requête en un vecteur de la même façon que les documents, puis en utilisant la similarité cosinus afin de sélectionner le document le plus proche de la requête. Pour implémenter le TF-IDF, nous avons utilisé la fonction correspondante de la bibliothèque scikit-learn.

### 1.3.2 Méthode probabiliste : BM25

Le modèle BM25 est une amélioration de la formule TF-IDF dans le cadre de la recherche d'information. Il s'agit d'un modèle de pondération utilisé dans les systèmes de recherche pour estimer la pertinence d'un document par rapport à une requête de l'utilisateur. Le "BM" signifie "Best Matching" et le "25" fait référence à la version spécifique du modèle.

Le score de pertinence dans BM25 pour un un document $d$ par rapport à une requête $q$ est donnée par le RSV (Retrieval Status Value) donné par la formule suivante :

$$\begin{aligned} RSV_{BM25}(d,q) = \sum_{j:t_j=r_j=1} & \log\left(\frac{N - df_{t_j} + 0.5}{df_{t_j} + 0.5}\right) \\ & \cdot \left(\frac{(k_1+1)\cdot tf_{t_j,d}}{k_1((1-b)+b\cdot\frac{L_d}{m}) + tf_{t_j,d}}\right) \\ & \cdot \left(\frac{(k_2+1)\cdot tf_{t_j,q}}{k_2 + tf_{t_j,q}}\right) \end{aligned}$$

où $L_d$ est la longueur du document $d$, $m$ est la longueur moyenne des documents dans la collection, et $N$ est le nombre total de documents

Le premier terme de la somme est le RSV d'un modèle MIB pour lequel la probabilité de présence d'un terme dans les documents pertinents et non pertinents est calculé d'après les lois 2-Poisson. Celui-ci est obtenu par une approximation asymptotique décrit par [Robertson and Walker, 1994].

Le deuxième terme de la somme permet de prendre en compte les occurrences des termes de la requête dans les documents, normalisées par la taille du document. Le paramètre $k_1$ contrôle l'importance de la fréquence et vaut par défault 1,2. Le paramètre $b$ contrôle l'importance de la longueur des documents et vaut par défault 0,75.

Le troisième terme permet la prise en compte de l'occurrence normalisée des termes de la requête dans la requête elle-même. Le paramètre $k_2$ permet de contrôler l'importance de cette prise en compte, et vaut par défault 1000.

Pour l'implémentation du BM25, nous avons utilisé le module python Rank-BM25 implémenté par [Brown, 2020]

### 1.3.3 Méthode linguistique : les n-grams

Les n-grams, définis comme des séquences continues de $n$ éléments issus d'un texte, offrent une approche complémentaire aux méthodes basées sur la fréquence des mots isolés, comme le TF-IDF, et les modèles probabilistes tels que le BM25. À la différence de ces dernières, qui analysent les mots de manière isolée, les n-grams permettent d'appréhender les séquences de mots, apportant ainsi une perspective additionnelle sur le contexte linguistique.

La génération des n-grams implique d'extraire, pour un $n$ donné, toutes les combinaisons possibles de $n$ éléments successifs dans un texte. Cette méthode ne se limite pas à l'importance de mots individuels (unigrams) mais s'étend à l'analyse des combinaisons et de la proximité des mots dans des séquences plus étendues (bigrams, trigrams, etc.), jusqu'à $n = 5$ dans notre cas.

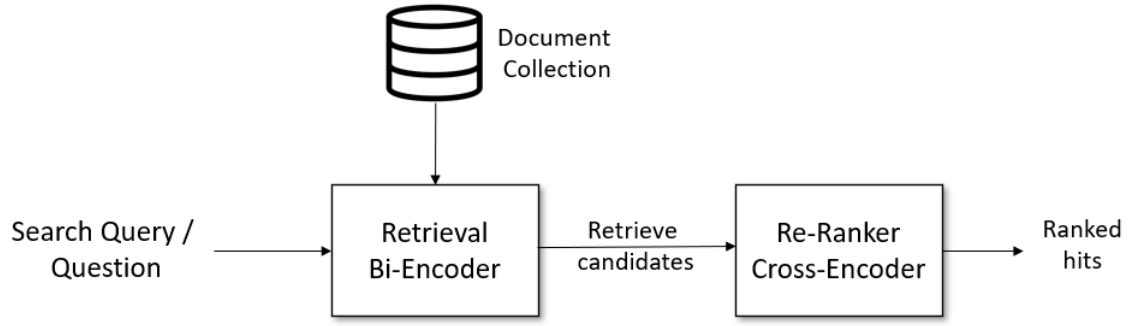
Pour l'indexation des documents, un index des n-grams est créé, associant chaque n-gram à sa fréquence dans le corpus. Pour rechercher des documents pertinents à partir d'une requête, cet index est utilisé afin d'identifier les documents comportant des n-grams similaires à ceux de la requête. Les scores de pertinence sont alors calculés en fonction de la fréquence des n-grams et ajustés pour accorder davantage d'importance aux n-grams de plus grande taille. Ceci est reflété par la formule $S_{ID} = \sum(f_{n\text{-gram},ID})^n$, où $f_{n\text{-gram},ID}$ est la fréquence du n-gram dans le document. Cette décision de valoriser les séquences de mots capturant des contextes plus spécifiques est basée sur l'hypothèse que les n-grams longs, susceptibles de capturer des expressions ou des contextes plus précis, ont un impact plus significatif sur le score de pertinence, facilitant ainsi une correspondance plus précise entre les requêtes des utilisateurs et les contenus des documents.

### 1.3.4 Méthodes basés sur les embeddings de paragraphes : BERT

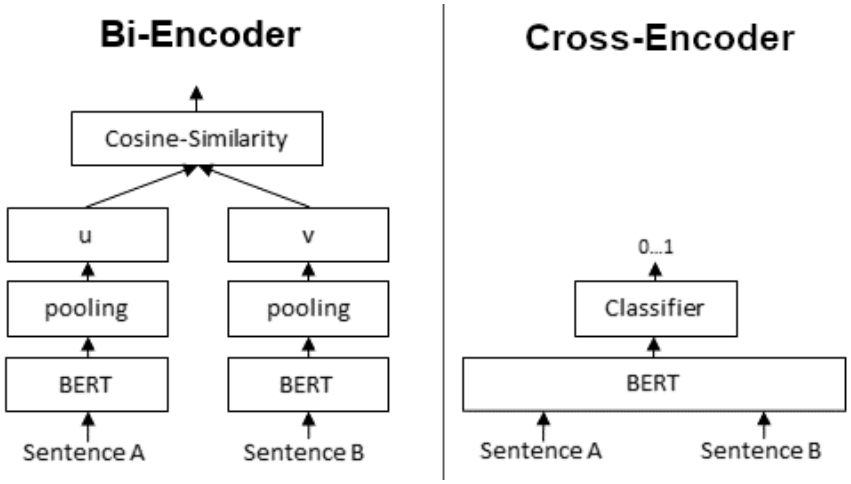
Nous avons ensuite utilisé des méthodes de deep learning pour transformer les paragraphes en embeddings afin de pouvoir effectuer une recherche à partir d'un calcul de similarité cosinus dans la base de données. Pour ce faire, nous avons utilisé le modèle SBERT de [Reimers and Gurevych, 2019] qui transforme un paragraphe en un vecteur de taille 384. Le modèle est basé sur BERT, un réseau de transformers entraîné pour des tâches telles que la réponse à des questions, la classification de phrases ou encore la régression pour des paires de phrases. Afin de transformer BERT en un réseau de type Bi-Encoder permettant la comparaison efficace d'embeddings, notamment par similarité cosinus, le réseau a été *fine-tuné* par un réseau siamois. Le réseau a été entraîné sur les données de SNLI et Multi-Genre NLI pour un total de 1 million de paires de phrases.

Nous utilisons ce modèle pour transformer les paragraphes en un embedding qui sera ensuite comparé avec la requête par une mesure de similarité cosinus.

### 1.3.5 Méthode basée sur les cross-encoder




Afin d'affiner les résultats obtenus à partir des embeddings précédents, nous avons ajouté une étape d'affinage du classement des documents les plus pertinents avec un cross-encoder. Les cross-encoders évaluent la pertinence entre une requête et un document de manière plus précise. Cependant, le modèle prend en entrée à la fois la requête et chaque document auquel il doit être comparé. Ce processus est donc bien plus coûteux qu'un calcul de similarité cosinus sur des embeddings calculés à l'avance. Afin d'accélérer la recherche, un premier classement est effectué à partir des embeddings obtenus avec le modèle présenté précédemment (Bi-encoder), avant de reclasser le top 50 des recommandations avec le Cross-encoder pour plus de précision. Cette approche optimise l'équilibre entre la précision de la recherche et l'efficacité computationnelle.




### 1.3.6 Utilisation d'embeddings obtenus avec l'API d'OpenAI

Enfin, nous avons comparé toutes les méthodes présentées précédemment avec l'utilisation de l'API d'OpenAI. Nous avons utilisé la fonction d'embedding d'OpenAI afin de transformer chaque document en embedding de dimension 3072. Le coût de l'utilisation de l'API pour les embeddings est raisonnable, de l'ordre de quelques centimes de dollars pour nos 10 heures de vidéo de test. La recherche s'effectue ensuite par simple similarité cosinus avec l'embedding de la requête.

# 2 Évaluation des algorithmes de recherche d'information

## 2.1 Création d'une base de données de tests

Nous avons constitué une base de données spécifique pour évaluer les algorithmes de recherche d'information, composée de vidéos de formation en français issues de YouTube et de la base de données de Klee. Cette base de test contient 16 vidéos, représentant un total de 10 heures et 41 minutes. Après le prétraitement, elle a été segmentée en 403 chunks dont la durée moyenne de chaque chunk est de 1 minute et 35 secondes.

Par la suite, nous indiquons des temps d'exécution pour les différentes étapes de l'algorithme. Il est à noté que ces temps d'exécution ont été mesurés sur un ordinateur portable d'étudiant, sans GPU. Cela signifie que ces temps sont plus indicatifs et pour faciliter la comparaison entre méthodes que des performances absolues. Par ailleurs, des étapes comme la transcription et l'embedding via OpenAI dépendent du temps de réponse des APIs concernées et ne sont pas directement liées à la puissance de calcul de l'ordinateur.

Tout d'abord, les temps de prétraitement de la base de données sont les suivants :

- Conversion des vidéos en audio : 6,19 minutes
- Transcription des audios : 119,88 minutes
- Découpage des transcriptions en chunks : 38,52 minutes

Pour les besoins des tests, 30 paires de questions-extraits ont été créées à la main en formulant une question pertinente pour un extrait sélectionné aléatoirement. L'objectif pour l'algorithme est donc de classer en premier l'extrait correspondant à la source de la question pour obtenir un score maximal. L'indicateur de qualité pour un modèle donné est le classement renvoyé de la bonne réponse. Afin d'agréger les 30 paires de test, nous utilisons le MRR (Mean Reciprocal Rank) défini ci-dessous. Plus le MRR est élevé, plus le modèle est précis.

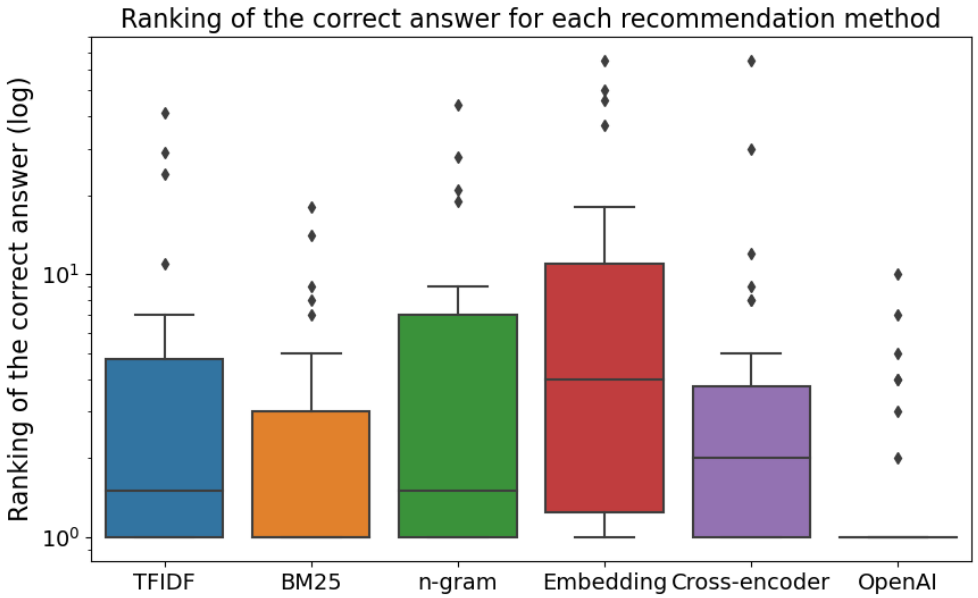
Le *Mean Reciprocal Rank* (MRR) est défini comme la moyenne de l'inverse des rangs de la première réponse correcte pour un ensemble de requêtes $Q$, formulé comme suit :

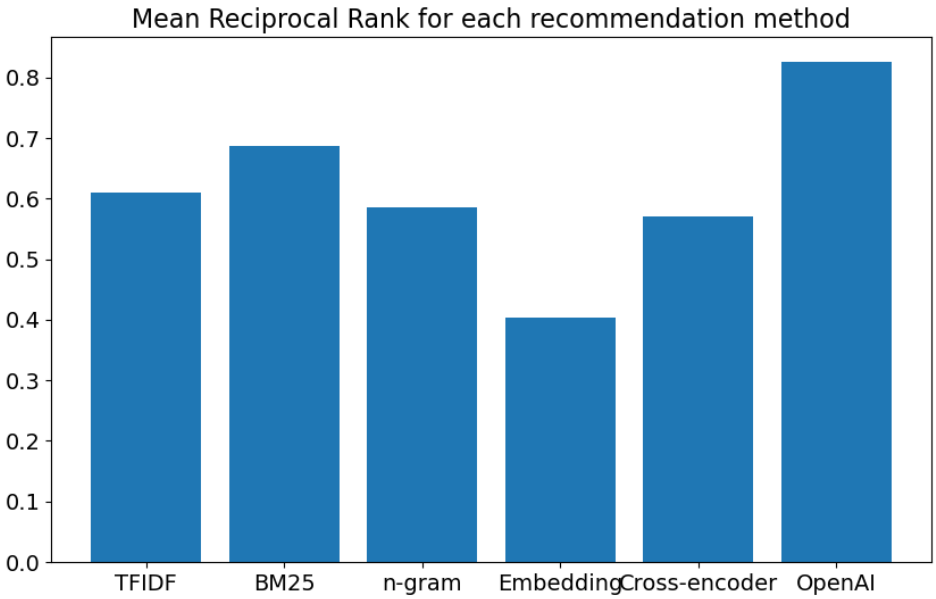
$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rang}_i}$$

où $|Q|$ représente le nombre total de requêtes, et $\text{rang}_i$ est le rang de la première bonne réponse pour la $i$-ème requête.

## 2.2 Résultats des comparaisons des différentes méthodes de recherche d'information

Voici ci-dessous les résultats des différentes méthodes utilisées, ainsi que les temps d'indexation de la base de données pour chaque méthode, et le temps de calcul moyen par recherche.






| **Méthode** | **Indexation/Embedding (s)** | **Recommandation (s)** |
|---|---|---|
| TFIDF | 0,126 | 0,0038 |
| BM25 | 0,105 | 0,0029 |
| n-gram | 1,961 | 0,0020 |
| Embedding | 41,22 | 0,079 |
| Cross-encoder | / | 15,75 |
| OpenAI | 166,1 | 1,179 |

On remarque tout d'abord que l'utilisation de l'API d'OpenAI peut largement être justifiée pour l'application de Klee étant donné qu'elle permet de retrouver la majorité du temps l'extrait adéquat comme première recommandation et surpasse ainsi largement les autres méthodes. Par ailleurs, nous remarquons que la méthode de l'embedding basé sur SBERT est la moins performante, et l'utilisation du cross-encoder pour affiner la recommandation ne permet pas de surpasser l'efficacité de la méthode plus ancienne du BM25. Enfin, notre implémentation simple du n-gram ne montre pas une grande efficacité et s'avère même moins performante que le TF-IDF. Nous remarquons enfin que les temps de recommandations sont, dans les conditions de test, toutes raisonnables, exceptée l'utilisation du cross-encoder qui peut s'avérer longue, notamment car le test a été effectué sur CPU.

Les bémols de l'utilisation d'OpenAI restent raisonnables. Le prix est faible pour la tâche d'embedding (environ 1 centime de dollar pour les 10h de vidéo test), et le temps de création des embeddings reste également raisonnable (moins de 3 minutes pour 10h de vidéo). Mais le temps de recommandation peut être important, notamment à cause de la taille importante des embeddings. Afin de dépasser cette limitation qui serait problématique au passage à l'échelle, nous pourrions utiliser une base de données vectorielle afin d'accélérer la recherche des embeddings les plus proches de l'embedding de la requête.

# Références

[Brown, 2020] Brown, D. (2020). Rank-BM25 : A Collection of BM25 Algorithms in Python.

[Guhr et al., 2021] Guhr, O., Schumann, A.-K., Bahrmann, F., and Böhme, H. J. (2021). Fullstop : Multilingual deep models for punctuation prediction.

[Radford et al., 2022] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision.

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert : Sentence embeddings using siamese bert-networks.

[Robertson and Walker, 1994] Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval.