# TalanSeeker : Application de l'architecture RAG pour la sélection de profils en conseil

Steve Bellart*, Arnaud Deleruyelle*

*Talan, 14 Rue Pergolèse, 75116 Paris
prenom.nom@talan.com,
https://www.talan.com/global/fr

**Résumé.** La sélection précise de profils pour répondre aux besoins spécifiques des clients est un défi majeur pour les sociétés de conseil. Cet article présente TalanSeeker, un système innovant basé sur l'architecture RAG (Retrieval Augmented Generation), conçu pour optimiser ce processus. L'architecture RAG combine la génération de langage naturel avec la récupération d'informations pertinentes, permettant une correspondance plus fine entre les compétences des consultants et les exigences des clients. Nous explorons l'efficacité de TalanSeeker dans le contexte de Talan, en évaluant sa capacité à améliorer la pertinence des profils sélectionnés et à réduire le temps de réponse. Les résultats préliminaires suggèrent que l'utilisation de l'architecture RAG peut significativement améliorer les processus de sélection de profils en conseil, même si certains problèmes subsistent.

## 1 Introduction

La sélection précise de profils pour répondre aux besoins spécifiques des clients est un enjeu majeur pour les sociétés de conseil. Avec l'augmentation du volume de données et la diversification des compétences disponibles, il devient essentiel d'adopter de nouvelles approches pour traiter et analyser ces informations efficacement. L'idée principale de l'architecture RAG est de représenter les documents et informations sous forme de vecteurs, qui serviront de base pour différents algorithmes comparatifs, permettant de mieux répondre aux diverses questions. Dans notre cas, cela permet notamment de retrouver certaines compétences spécifiques dans un grand nombre de CV, facilitant ainsi le choix des profils pour répondre à un besoin.

Introduite par Lewis et al. (2020), l'architecture RAG intègre deux modules principaux : un module de récupération qui extrait des documents ou des informations pertinentes d'une base de connaissances, et un module de génération qui produit des réponses en se basant sur ces extractions. Initialement, cette architecture s'appuyait sur des techniques d'embedding traditionnelles pour représenter les documents et les requêtes. Cependant, avec l'avènement des grands modèles de langage (LLM) tels que GPT-3 par Brown (2020) et BERT par Kenton et Toutanova (2019), il est désormais possible d'exploiter les embeddings issus de ces modèles pour améliorer la performance des systèmes RAG.

Les embeddings dérivés des LLM capturent des représentations sémantiques riches et contextuelles, ce qui permet une meilleure correspondance entre les requêtes et les sources

lors de la phase de récupération des éléments. Par exemple, Khattab et Zaharia (2020) ont proposé ColBERT, un modèle qui utilise des embeddings contextuels pour la récupération de passages, améliorant ainsi la précision des systèmes de question-réponse. De plus, les travaux de Gao et al. (2021) sur les modèles de contraste ont démontré que l'entraînement contrastif des LLM peut produire des embeddings de haute qualité pour la recherche sémantique. De façon plus générale, les approches utilisant des embeddings issus de l'essor des LLM se multiplient, comme le montre Gao et al. (2023).

D'autres approches analogues ont été proposées : par exemple Guu et al. (2020) ont développé REALM, qui incorpore un mécanisme de récupération directement dans le processus de pré-entraînement du modèle de langage. De même, Izacard et Grave (2020) ont présenté Fusion-in-Decoder, une architecture qui fusionne les informations de plusieurs sources pour améliorer la génération de réponses dans le contexte des questions ouvertes.

Plusieurs outils exploitant l'IA visent à simplifier la sélection de profils et la gestion des compétences, comme Napta (2024) qui se concentrent sur la planification des ressources et la création de CV formatés. Ces approches atteignent leurs limites face à des données non structurées comme des CV bruts ou des analyses contextuelles complexes.

Cet article présente TalanSeeker, un système basé sur une architecture RAG adapté à la sélection de profils dans le secteur du conseil. Nous proposons de vectoriser des morceaux de CV (*chunks*) selon diverses stratégies, puis de vectoriser une expression d'un besoin afin de trouver les CV les plus pertinents pour y répondre. Nous présentons dans un premier temps l'architecture de TalanSeeker en précisant comment elle est déployée dans l'entreprise Talan. Nous présentons ensuite diverses mesures que nous avons faites, comme l'importance de la stratégie de morcelage des CV, les problèmes d'équité détectés ainsi qu'une évaluation des embedders utilisés.

# 2 Architecture

## 2.1 Structure générale

TalanSeeker est un outil dont le but est de trouver les profils les plus pertinents parmi ceux connus via une base de données de CV (en formats divers : .docx, .pptx, .pdf...), en fonction d'un besoin exprimé par l'utilisateur. Le fonctionnement général de l'outil est décrit dans le schéma 1. TalanSeeker est alimenté par trois canaux distincts :

— **CV bruts :** Un stockage des CV des personnes que l'outil doit pouvoir remonter suite à une requête si le profil répond à la demande.
— **Des informations générales :** D'autres informations que celles présentes dans les CV peuvent être données à TalanSeeker pour améliorer son efficacité.
— **L'expression du besoin :** Le besoin à satisfaire, présenté comme une demande en langage naturel.

Hormis la mise à jour des informations générales, qui ne sera pas développée ici car propre à la version industrielle de l'outil, il y a donc deux processus principaux qui font fonctionner TalanSeeker : l'ajout d'un CV à prendre en compte et l'expression d'un besoin auquel il faut répondre.

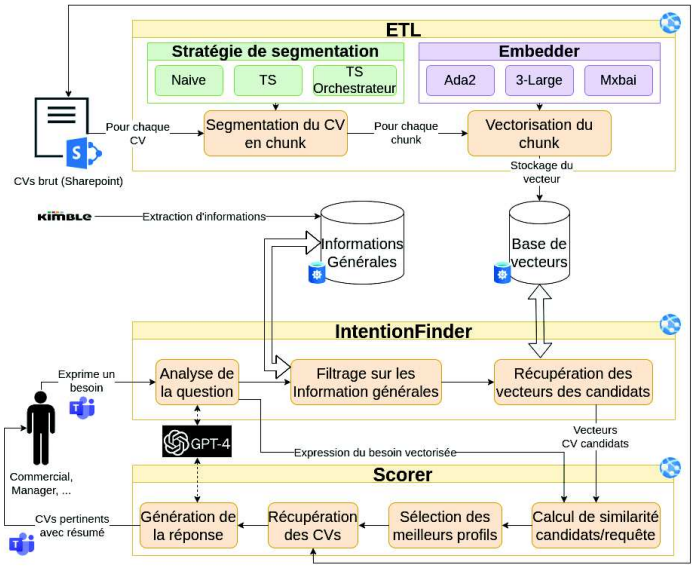


FIG. 1 – Schéma général de TalanSeeker

Pour la mise à jour des CV, notre stratégie repose sur deux étapes complémentaires qui permettent d'uniformiser le format des CV et d'en extraire les informations les plus pertinentes, à fournir à l'algorithme de comparaison sémantique

1. **ETL :** Le module ETL (Extract, Transform, Load) joue un rôle central en permettant de traiter en entrée tout type de format de CV et de produire une représentation uniforme et standardisée, mettant en valeur les compétences du collaborateur. Pour atteindre cet objectif, ce module s'appuie sur des modèles de langage capables de traiter directement les fichiers binaires des CV. Ces modèles génèrent une version textuelle des CV, qui est ensuite sauvegardée dans notre système de persistance des données.
2. **Stratégie de division du CV :** Une fois la sortie de l'ETL obtenue, nous devons segmenter ce CV textuel en plusieurs morceaux (*chunks*) afin de pouvoir l'encoder en plusieurs vecteurs. De nombreuses stratégies ont été testées pour réaliser ce découpage, en plus de celle actuellement retenue pour la version déployée de TalanSeeker (stratégie TS sur le schéma 1), car cela a un impact sur la qualité de la sélection des profils. Une fois le CV segmenté, il faut désormais vectoriser chacun des *chunks*, et le choix de l'embedder a aussi son importance, comme détaillé dans la section 3. Les vecteurs sont ensuite stockés dans une base de données, prêts à l'emploi.

Pour répondre à un besoin exprimé par un utilisateur, TalanSeeker va générer sa réponse en deux temps :

1. **IntentionFinder :** Tout d'abord, il y a une analyse de la requête exprimée en langage naturel. Cela est fait en utilisant un LLM afin d'extraire toutes les informations de la requête, même si elles sont mal exprimées. L'analyse cherche à extraire, entre autres, les compétences, les formations ou encore l'expérience attendue des profils répondant

au besoin, mais aussi des informations sur la localisation ou le début de la mission. Ce sont notamment ces dernières informations qui permettent de filtrer des candidats incompatible administrativement permettant d'éviter des calculs inutiles sur les vecteurs. Enfin, pour les candidats ayant passé ce filtre, leurs vecteurs issus de leur CV, ainsi que la requête de l'utilisateur, analysée et également vectorisée, sont envoyés au *Scorer*.

2. **Scorer :** Le *Scorer* calcule la similarité via le produit scalaire entre le vecteur exprimant le besoin de l'utilisateur et les vecteurs représentant les morceaux de CV des candidats ayant passé le filtre. Les CV dont la moyenne de similarité de leurs *chunks* avec le vecteur de la requête est la plus élevée sont les profils retournés par TalanSeeker. Une réponse finale est générée en se basant sur un résumé des CV sélectionnés, leurs liens avec le besoin ainsi que les liens pour télécharger les CV d'origine.

Une fois le besoin satisfait, l'utilisateur peut choisir de préciser sa demande via une prise en compte du contexte des échanges précédents ou se dire satisfait, clôturant la demande actuelle.

### 2.2 Déploiement et accès

La version déployée au sein de Talan est celle présentée dans la vidéo. TalanSeeker propose une interface intégrée directement dans Teams de l'entreprise via un bot avec lequel on peut exprimer ses besoins en langage naturel. Le reste de l'application est hébergé sur un espace Azure de l'entreprise où les CV sont également stockés conformément au RGPD (environ 800 actuellement). Il est intéressant de noter que l'architecture RAG de TalanSeeker permet d'éviter un apprentissage d'IA sur les CV, ce qui facilite le respect du droit à l'oubli. Les modèles d'IA utilisés dans cette version sont des modèles disponibles sur Azure. L'embedder sélectionné est Ada2 et le LLM est GPT4-o-mini. Le code reste néanmoins disponible sur Git [1] avec des CV synthétiques à disposition.

## 3 Etude de la solution

### 3.1 Présentation des CV synthéthiques

Afin d'évaluer les performances de notre solution, nous avons dû concevoir un ensemble de tests précis pour lequel nous disposons, en amont, d'une connaissance des meilleurs profils. Ce jeu de données cible des catégories de métiers spécifiques, en tenant compte de profils ayant des niveaux d'expérience plus ou moins variés, selon les exigences propres à chaque métier.

Chaque CV est représenté selon une mise en forme spécifique : une description concise du profil, une liste de compétences techniques, les formations académiques suivies, ainsi que la liste des expériences professionnelles. Cette mise en forme correspond à celle produite par l'ETL de notre solution et permet, au-delà d'améliorer la visibilité du document, d'effectuer une séparation plus efficace en chunks ce qui aboutit à de meilleures performances globales.

En ce qui concerne la quantité de CV, nous disposons de 6 CV par métier ciblé, avec des niveaux d'expérience graduels allant de profils juniors à seniors. Cette répartition nous permet de tester notre programme non seulement sur différents métiers, mais également en fonction de la séniorité des profils.

1. Dépôt : `https://github.com/TalanRecherche/Talanseeker-Backend`

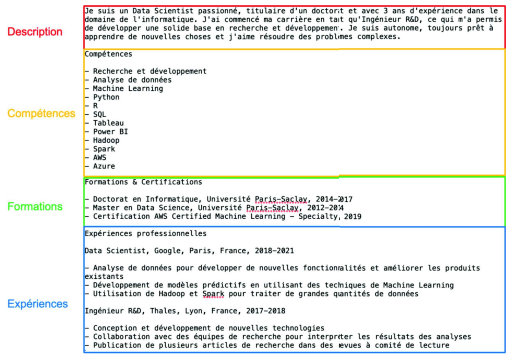


FIG. 2 – CV généré artificiellement avec découpage intuitif en 4 sections.

## 3.2 Présentation de la liste de questions

La liste de questions cible trois thématiques principales : (1) 12 questions portant sur l'obtention de profils correspondant à des métiers spécifiques, (2) 12 questions évaluant l'adéquation entre un métier et un niveau d'expérience, et enfin (3) 8 questions concernant la correspondance entre un métier, une expérience précise et une entreprise donnée.

Pour chacune des questions, nous évaluons le classement des meilleures réponses proposées en fonction du nombre total de bonnes réponses possibles. Par exemple, si nous demandons à l'algorithme de fournir les CV de Data Scientists et que notre base en contient 6, nous récupérons les 6 premières réponses de l'algorithme. Nous calculons ensuite un ratio de bonnes réponses retournées parmi ces 6 profils. Voici un rapide extrait des questions de notre liste :

**Catégorie "recherche d'un métier"**

— Je veux un Data Scientist.
— Je veux un développeur Web.

**Catégorie "seniorité"**

— Je veux un Business Analyst expérimenté.
— Je veux un expert en sécurité informatique avec peu d'expériences.

**Catégorie "recherche d'une entreprise"**

— Je veux un data scientist ayant une expérience chez Capgemini.
— Je veux un/e travailleur/se dans les ressources humaines ayant une expérience chez L'Oréal.

## 3.3 Etude sur l'impact des chunks, et du choix de l'embedder

Cette section présente une comparaison ciblée des améliorations apportées à notre stratégie de RAG. Nous avons testé 3 vectoriseurs différents, et nous étudions l'impact de la stratégie de découpage des documents. Dans la méthode naïve, les documents sont segmentés tous les 256 caractères, en forçant une coupure après un saut de ligne. Notre stratégie (noté TS) quant à elle, divise chaque document en quatre en suivant une logique précise. (section 3.1)

En complément, nous avons exploré l'ajout d'un orchestrateur qui, en identifiant le type de requête, pourrait appliquer une pondération spécifique pour raffiner les résultats. Cette optimi-

| Embedder | Catégorie | Découpage naif | TS | TS + orchestrateur |
|---|---|---|---|---|
| embedding-ada-002 (OpenAI) | Métiers | 0.861 | 0.917 | 0.944 |
| | Séniorité | 0.708 | 0.750 | 0.833 |
| | Entreprise | 0.167 | 0.541 | 0.541 |
| embedding-3-large (OpenAI) | Métiers | 0.861 | 0.944 | 0.972 |
| | Séniorité | 0.667 | 0.750 | 0.833 |
| | Entreprise | 0.167 | 0.416 | 0.541 |
| mxbai-embed-large (Meta) | Métiers | 0.861 | 0.889 | 0.944 |
| | Séniorité | 0.625 | 0.667 | 0.708 |
| | Entreprise | 0.167 | 0.541 | 0.541 |

TAB. 1 – Précision de différents RAG pour trois corpus de questions ciblés, en fonction de la stratégie de division (chunk) du document. Le tableau montre un découpage naif, celui proposé par notre solution (TS) ainsi qu'une variante pour la gestion de la pondération optimale (orchestrateur).

sation est possible car le nombre de chunks est toujours constant. Pour évaluer cet apport, une optimisation des paramètres est réalisée par une recherche locale itérative.

Les résultats présentés dans le tableau 1 soulignent des hausses significatives des performances de l'algorithme, indépendamment du vectoriseur utilisé. Une amélioration minimale de 5 points est observée pour les requêtes portant sur la recherche d'un métier ou d'un niveau d'expérience précis. Ces gains peuvent atteindre jusqu'à 30 points dans le cas de requêtes spécifiques liées à une expérience professionnelle, confirmant l'efficacité de notre méthode.

Enfin, l'utilisation d'un orchestrateur contribue à une amélioration globale des performances. Cette approche permet d'adapter les pondérations en fonction du type de requête, renforçant ainsi la précision des résultats. Toutefois, il est important de souligner que ces performances dépendent également d'un autre modèle de langage, devant comprendre avec précision la nature de la question posée afin d'ajuster efficacement les paramètres en conséquence.

## 3.4 Analyse éthique et equité

Cette section traite des biais liés à la parité homme-femme dans les sélections de l'algorithme. L'expérience repose sur la génération de paires de CV identiques, différant uniquement par le genre de l'individu. Les résultats présentés dans le tableau proviennent d'une moyenne sur de nombreuses requêtes non genrées dont le but est d'identifier les "Data Scientists". L'algorithme considère ici le découpage des documents en quatre chunks : description générale, compétences, formations et expériences professionnelles.

Les observations montrent que les sections "formations" et "compétences" restent identiques pour chaque paire de CV, le genre n'affectant pas la manière dont ces informations sont présentées. Cependant, pour les sections "description générale" et "expériences professionnelles", des différences apparaissent dans les intitulés de postes, les adjectifs employés et d'autres éléments textuels, conférant un léger avantage aux profils masculins. Ces résultats, reproduits sur diverses requêtes, révèlent une tendance générale à prioriser, bien que modestement, les hommes pour des compétences équivalentes.

| Métier | Sexe | C.Desc | C.Comp | C.Form | C.Expé |
|---|---|---|---|---|---|
| Data Scientist | H | 0.507 | 0.375 | 0.338 | 0.449 |
| Data Scientist | F | 0.490 | 0.375 | 0.338 | 0.451 |
| ML Ingénieur | H | 0.414 | 0.437 | 0.384 | 0.410 |
| ML Ingénieur | F | 0.402 | 0.437 | 0.384 | 0.411 |
| Ch. de recrutement | H | 0.158 | 0.166 | 0.148 | 0.200 |
| Ch. de recrutement | F | 0.149 | 0.166 | 0.148 | 0.199 |

TAB. 2 – Moyenne des scores de proximité de chaque chunk pour des requêtes portant sur les "Data Scientists", en considérant des paires de CV identiques pour chaque métier, à l'exception du genre de l'individu. Embedding-3-large (OpenAI) est utilisé pour la vectorisation.

Pour remédier à ces biais, nous avons exploré deux alternatives. La première approche consiste à modifier systématiquement les documents pour qu'ils soient rédigés au masculin. La seconde repose sur une traduction des documents en anglais, une langue plus neutre. Cependant, cette traduction n'est pas entièrement déterministe, ce qui peut introduire de légères variations dans les documents, influençant tout de même les résultats de l'algorithme.

# 4 Conclusion et perspective

Les résultats de notre étude montrent qu'une approche par recherche sémantique comme TalanSeeker est pertinente pour de l'affectation de profils. Contrairement aux méthodes traditionnelles de RAG, TalanSeeker se distingue par trois points clés : un ETL performant capable de convertir n'importe quel document en un format textuel compréhensible pour les modèles de langage, un découpage optimisé des documents permettant des gains significatifs dans la recherche de profils, et une gestion éthique du module de score garantissant que seules les compétences sont prises en compte, excluant toute autre considération.

En parallèle, l'application se veut ergonomique avec une intégration directe dans des plateformes largement utilisées comme Teams, tout en incluant une gestion des droits utilisateurs et des optimisations de calcul pour fournir des réponses en une dizaine de secondes.

Bien que TalanSeeker offre des performances significatives, nos résultats montrent une variabilité liée au choix de l'embedder ainsi que des dilemmes éthiques sur la gestion des biais de genre. Pour y faire face, nous forçons l'algorithme à retourner une liste élargie de profils afin d'éviter des erreurs lors de la comparaison des documents. De plus, un mélange sur l'ordre des profils est effectué au niveau de l'enrichissement du contexte final, donné au modèle de langage. Cela nous garantit un traitement identique pour chaque profil.

Après ces premiers résultats, plusieurs pistes sont envisagées pour enrichir TalanSeeker :

1. **Stabilisation de l'orchestrateur** : Amélioration de la classification des types de requêtes en entrée pour la rendre plus souple et précise.
2. **Gestion des biais de genre** : Les solutions proposées pour la gestion du genre fonctionnent en l'état, mais elles ne sont pas idéales. Nous souhaitons explorer d'autres alternatives plutôt que d'avoir des modèles raisonnant que sur des profils masculins.

3. **Optimisation des temps de réponse** : Développement en cours d'une intégration sur Hana Vector Engine pour accélérer les calculs vectoriels.

Ces travaux futurs visent à consolider TalanSeeker comme une solution robuste, éthique et performante pour la gestion des ressources humaines basée sur la recherche sémantique.

## Références

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv :2005.14165*.

Gao, T., X. Yao, et D. Chen (2021). Simcse : Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv :2104.08821*.

Gao, Y., Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, et H. Wang (2023). Retrieval-augmented generation for large language models : A survey. *arXiv preprint arXiv :2312.10997*.

Guu, K., K. Lee, Z. Tung, P. Pasupat, et M. Chang (2020). Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR.

Izacard, G. et E. Grave (2020). Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv :2007.01282*.

Kenton, J. D. M.-W. C. et L. K. Toutanova (2019). Bert : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, Volume 1, pp. 2. Minneapolis, Minnesota.

Khattab, O. et M. Zaharia (2020). Colbert : Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 39–48.

Lewis, P., E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems 33*, 9459–9474.

Napta (2024). Napta : Optimizing resource management and staffing. Consulté le 20/11/2024.

## Summary

Precise profile selection to meet clients' specific needs is a major challenge for consulting firms. This paper introduces TalanSeeker, an innovative system based on the RAG (Retrieval-Augmented Generation) architecture, designed to optimize this process. The RAG architecture combines natural language generation with the retrieval of relevant information, allowing for a finer match between consultants' skills and clients' requirements. We explore the effectiveness of TalanSeeker in the context of Talan, evaluating its ability to improve the relevance of selected profiles and reduce response time. Preliminary results suggest that using the RAG architecture can significantly enhance profile selection processes in consulting, paving the way for new AI-based approaches in recruitment.