Chapitre 7 : Utiliser les graphes de connaissances pour « augmenter » les LLMs

Les graphes de connaissances ou bases de connaissances sont un format particulièrement utile pour stocker et connecter aussi bien des informations générales que des informations d'un domaine de spécialité. Ils constituent une façon structurée de représenter la réalité, des fait et leurs relations. Un premier constat est que la combinaison de données structurées et non structurées peut s'avérer très utile pour de nombreuses tâches de NLP, et c'est d'autant plus vrai à l'ère des modèles génératifs car on peut s'en servir pour vérifier la factualité des réponses ou augmenter leurs capacités. Utiliser les graphes de connaissances est ainsi une approche prometteuse pour augmenter les connaissances et les capacités de raisonnement des LLM, ce qui est d'ailleurs la raison principale pour laquelle l'ingénierie des connaissances a récemment fait son « come back » sur le devant de la scène de la recherche en NLP..

Les graphes de connaissance connaissent en effet un intérêt croissant dans la littérature. On ne comptait qu'une quinzaine d'articles sur la question en 2025, plus de 150 en 2020. Le plus souvent, le champs applicatif de ces travaux porte sur le domaine médical.

Plusieurs axes prometteurs ont été explorés pour procéder à la bonne intégration des graphes de connaissances avec les LLM. L'interaction des deux domaines est au final assez naturelle.
* La génération de textes à partir de données structurées, et notamment à partir de graphes était un sujet important du NLP avant l'arrivée des LLMs. On entraînait des modèles de type BERT à rédiger du texte à partir de triplets de graphe de connaissances ce qui pouvait être très utiles pour automatiser la rédaction dans des domaines de spécialité[1]. Évidemment l'arrivée des LLMs a rendu ces travaux en grande partir caduques.
* On peut suivant la même idée, au moment de l'inférence extraire certains éléments pertinents d'un graphe de connaissance et fournir cette donnée structurée au LLM pour compléter sa réponse. C'est le cas d'usage le plus exploré, en s'inspirant de toutes les autres approches qui se proposent de compléter les informations du modèle au moment de l'inférence.
* L'embedding de graphes de connaissance en des vecteurs denses capturant les entités et leurs relations. Ces vecteurs sont ensuite utilisés lors de l'entraînement ou de l'inférence du LLM pour lui permettre de tirer parti des connaissances contenues dans la base. L'inconvénient est que cette approche nécessite un grand travail mathématique à chaque fois qu'on veut la mettre en œuvre sur un modèle et avec une base de connaissances particuliers. De plus ce n'est pas du plug and play. De plus rien ne garantit que cette approche soit plus efficace qu'un fine tuning classique à partir de textes bruts.
* On peut augmenter le corpus d'entraînement des LLMs avec des triplets issus de graphes de connaissance. Cela permettrait d'améliorer la compréhension du LLM des données structurées mais présuppose que cet entraînement à reconstituer des triplets sera bénéfique aux performances du modèle. Il reste à démontrer que cette approche est plus pertinente que d'entraîner le modèle avec le texte de base dont a été issu le triplet.
* On peut explicitement faire du multi task learning entraînant les LLMs à mieux comprendre les KG et savoir les utiliser. C'est peut être plus ou moins fait pour les plus gros modèles (qui sait?)

1 https://aclanthology.org/2020.tacl-1.38.pdf

mais nécessite en tout cas un corpus spécialisé. On espère que cet entraînement spécifique donnerait aux LLMs de meilleures capacités de raisonnement sur les données issues de graphes de connaissance. Notamment la capacité de raisonner sur plusieurs chemins pour répondre à des questions de type multi hop TODO (path ranking), du raisonnement logique permettant de faire de l'inférence sur les nœuds (si (Hobbit, vivent_dans, la Compté) et (Frodon, est_un, hobbit), alors (Frodon, vit_dans, la Compté).

On ne mentionnera pas dans ce chapitre l'usage des LLMs pour construire des graphes de connaissances de façon automatique (end to end) car cela a été traité de façon transversale notamment dans le chapitre 3. Un axe de recherche important est l'évaluation des graphes obtenus automatiquement et leur comparaison avec un graphe qui aurait été construit à partir de l'expertise humaine.

Les graphes de connaissance ont depuis leur apparition et depuis que l'on dispose de manières efficaces pour les construire, toujours été utilisés comme « béquille » pour venir soutenir d'autres tâches de NLP. Par exemple dans la tâche du résumé de texte extractif, disposer des relations sous forme de graphe permet de repérer les entités les plus centrales et de sélectionner les morceaux de texte qu'il est important de retranscrire.

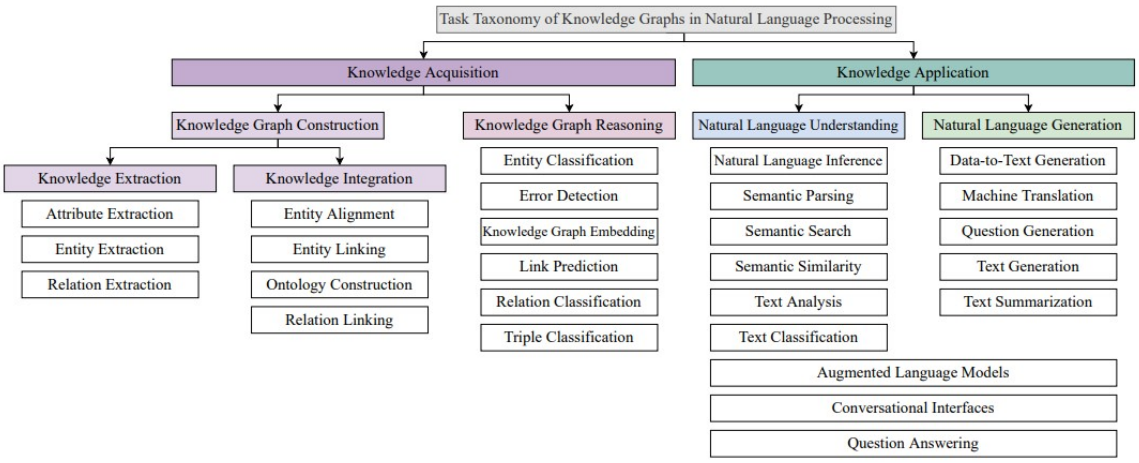


Figure 1: Taxonomy of tasks in the literature on KGs in NLP.

Les Graphes de connaissances peuvent notamment être employés pour le Question Answering. On appelle KBQA la variante de la tâche spécifiquement intéressée par l'interrogation de KB.[2] L'objectif de ce cours sera d'étudier l'emploi des graphes de connaissances pour venir en aide à d'autres tâches de NLP et notamment celles qui impliquent des LLMs. On se focalisera notamment sur l'approche dite « GraphRAG » qui n'est rien d'autre que l'incorporation de fragments d'un graphe de connaissances dans la fenêtre de contexte d'un LLM pour répondre à une question en langage naturel.

2 https://aclanthology.org/2022.aacl-main.46.pdf pour l illustration

I) Les différents usages des graphes de connaissance comme soutien à des tâches de NLP

Les graphes de connaissances sont très utiles pour venir épauler des tâches de NLP ou de data science pour lesquelles l'apport d'informations ou de précisions sémantiques est importante, notamment quand ces données sont fortement connectées. Ils peuvent servir aussi bien à stocker qu'à fournir l'information.

a) représenter un domaine de spécialité

Les graphes de connaissance, comme les ontologies, ont souvent été utilisés pour représenter les connaissances d'un domaine de spécialité. Le cas d'école est évidemment le domaine médical. Une fois les relations taxonomiques du graphe de connaissance établies, on peut alors identifier tous les concepts spécifiques comme les maladies, les organes touchés, les médicaments… Avoir un graphe permet (par détection de communauté ou clustering) de proposer des emplois de molécules pour soigner d'autres pathologies. L'application de règles permet aussi de repérer les contre indications majeures. Enfin, stocker toute cette connaissance sous forme de graphe la rend facilement requêtable. À noter que la frontière avec une ontologie est finalement assez mince dans le cas où le type des relations du graphe de connaissance sont arrêtés à l'avance.

Mais il n'y a pas que les domaines « traditionnels » qui peuvent bénéficier de la mise en forme sous forme de graphe de connaissance. N'importe quel domaine peut constituer la source d'un graphe nourri de données non structurées. Il faut pour cela définir clairement le schéma du graphe que l'on veut obtenir à la fin, c'est à dire quelles données extraire de ses sources (documents structurés ou non, leur métadonnées…) . Ensuite créer les composants qui vont extraire ces informations et nourrir le graphe.

Property Name
condition
background
motivation
leads_to
review
introduces
improves
resolves
argues
produces
supports
not support
summary
purpose-behavior
uses
basis
guides
elaboration

Même des domaines plus « exotiques » peuvent donner des graphes utiles et exploitables. Pour prendre l'exemple du domaine académique, l'explosion du nombre de publications (arXiv…) rend difficile la réalisation d'états de l'art complets. ils est possible de concevoir un graphe utile permettant de trouver plus facilement les articles qui sont utiles par rapport au sujet d'intérêt. Bien que de nombreux graphes archiv ont été créés, ils ne contenaient souvent que des métadonnées basiques (e.g., title, author, institution, keywords, issues, and publisher) qui ne prétait pas assez attention au sémantisme fin du contenu des articles. L'utilisation d'une ontologie des concepts de la recherche scientifique permet de mieux capturer leur spécificité[3]. Chaque phrase importante de l'article est associée à une étape clé de la démarche scientifique (cextrait du research goal, des arguments, illustrés par tel exemple…) Cela avait été fait par un lourd travail d'annotation et des systèmes de règmes mais pourrait très bien être réalisé par LLM ou par des systèmes de classification plus simples.
Ce graphe de connaissance permet d'identifier immédiatement les sections importantes des articles (conclusions, introductions…) et de faciliter la navigation dans une grande masse d'articles (comparer toutes les conclusions). On pourrait aller bien plus loin que le travail initié et se servir de cette idée pour rassembler par similarité des articles qui ont des conclusions similaires, avancent des arguments contraires… et développer un outil d'analyse comparée d'articles rafinnant ainsi la constitution d'états de l'art. Juste des idées si vous vous ennuyez le week end.

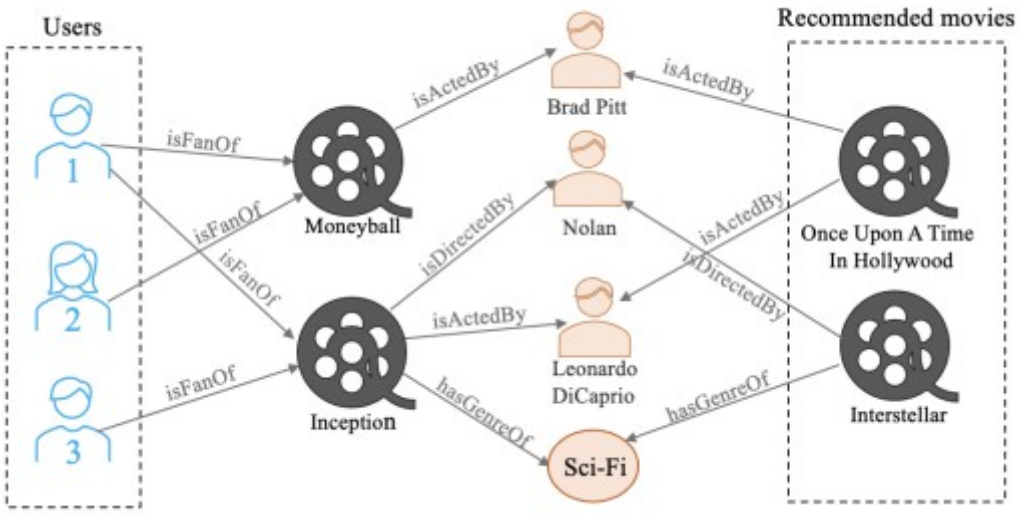
b) Les systèmes de recommandations

Avec le développement du big data, la quantité de données disponibles a explosé de sorte qu'il devient difficile de se repérer dans la masse d'information. D'où le travail sur les moteurs de recherche, mais aussi (la réciproque) sur les systèmes de recommandation. L'idée est de proposer à un utilisateur uniquement des contenus qui correspondent à ses goûts et à ses intérêts, (le plus souvent avec l'idée de lui vendre quelque chose à la fin…) . Ce genre de technologie est à la base des systèmes de personnalisation des publicités sur le web, des suggestions d'abonnement sur les réseaux sociaux ou encore de série à voir sur des plateformes comme Netflix

Parmi les méthodes pour construire des systèmes de recommandation, on peut distinguer :
* des méthodes basées sur le contenu lui même : sont identifiés les features principales de ce contenu (description de la série…). À partir des interactions précédentes de l'utilisateur, un modèle de machine learning apprend à distinguer ce qui peut l'intéresser ou non. On utilise des méthodes comme le calcul de corrélation, la similarité, des méthodes de classification… . Il faut toutefois bien identifier les features pour que cela fonctionne correctement.
* Les méthodes basées sur un filtrage collaboratif (Colaborative Filtering ou CF). L'idée est de se baser sur les interactions globales des utilisateurs et on déduit ce qui va intéresser une personne à partir de ce qui a intéressé des personnes avec un profil similaire. On utilise pour cela des techniques d'extraction de feature et aussi du clustering (des profils utilisateurs). Cette approche pose toutefois des problèmes de confidentialité sur les données utilisateurs.

3 https://ceur-ws.org/Vol-3304/paper03.pdf

* Le mieux est bien sur d'utiliser des solutions hybrides qui combinent le meilleur des deux mondes. La plupart du temps ces solutions hybrides reposent sur l'usage de graphes de connaissance.



Représenter les données sous forme de graphe permet de voir plus de connexions et de retrouver des connexions entre les objets. Par exemple pour le cas d'un système de recommandation de vidéos [4] les informations sur le film (genre, acteurs, réalisateur) permettent de mieux cerner les goûts de l'utilisateur et lui proposer du contenu proche thématiquement, ou qu'a apprécié un utilisateur avec les mêmes patterns.

Un des avantages de ces méthodes à base de graphes est qu'elles maintiennent une très bonne explicabilité. Le travail de feature extraction est visible dans le choix de la structure du graphe et de ces attributs, mais de ce fait on est capable d'extraire les patterns qui ont conduit à la proposition. Cela permet aussi de limiter la difficulté à recommander quelque chose aux nouveaux utilisateurs qui n'ont interagi qu'avec très peu de contenu puisque toute l'information est interconnectée. Idem quand on ajoute un nouveau produit qui peut être recommandé aux utilisateurs ayant interagi avec des contenus similaires.

c) la recherche documentaire / booster la recherche d'information

Les graphes de connaissance ont également souvent été employés comme support à des systèmes de recherche d'information ou de Question Answering (bien que souvent on puisse ramener ce second cas au premier). Si la plupart des systèmes de question answering se basent sur des données textuelles, les relations sémantiques présentes dans les textes peuvent être également utilisées.

Une des difficultés principales de la recherche documentaire est de devoir naviguer dans une très grande quantité de données non structurée. Si la recherche sémantique est imprécise (vecteurs non adaptés ou autre…) le résultat peut vite se dégrader. Utiliser des graphes structurés peut être une façon efficace de limiter cela.

* On peut se servir des graphes comme d'un outil de filtrage. En identifiant les entités présentes dans les contenus et dans la requête de l'utilisateur, on peut uniquement sélectionner une partie des

4 https://arxiv.org/pdf/2303.13948

documents pertinents et ainsi améliorer la précision de la recherche sémantique classique en réduisant l'espace de recherche.
* On peut aussi utiliser des graphes au moment de l'indexation dans la base. Il est possible de construire un graphe à partir des documents, parfois quand ils ont un vrai rapport entre eux (technique dite du « parent document »[5] qui rapprochent les textes extraits du même document long, comme par exemple un fichier pdf de plusieurs centaines de pages) ou ont été rapprochés par des techniques de topic modeling. Un peu selon la même philosophie d'indexation des vecteurs dans des algorithmes comme HNSW, cette structuration de la base documentaire comme arbre accélère la recherche et permet de tirer partie des liens existant entre les documents eux-mêmes.

Ce que l'on appelle recherche sémantique vise à aller au-delà de la simple correspondance littérale des termes pour comprendre l'intention et le contexte de la requête. Cette notion de compréhension de l'intention de l'utilisateur (user intent) est essentielle et sur cet aspect également les graphes peuvent aider. Contrairement aux approches traditionnelles basées sur la recherche par mots-clés, ils permettent d'intégrer des relations sémantiques entre les concepts, ce qui permet d'utiliser des techniques d'expansion de requêtes à partir des concepts synonymes.
Grâce aux graphes, il devient possible de mieux interpréter l'intention de l'utilisateur en reliant des termes associés et en tenant compte du contexte de la recherche. Par exemple, des graphes comme ConceptNet, qui modélisent les relations entre concepts du quotidien, ou le Microsoft Academic Graph, qui cartographie les liens entre publications scientifiques, permettent d'orienter les réponses en exploitant ces connexions.

Prenons un exemple avec Le Seigneur des Anneaux et une recherche dans une base de connaissances sur l'univers de Tolkien.
Requête initiale : "personnages du Gondor"
Sans expansion, la recherche pourrait simplement retourner une liste brute de personnages directement associés au Gondor, comme Aragorn ou Denethor.
En utilisant un graphe de connaissances (par exemple, un graphe basé sur les relations entre les personnages et les lieux de la Terre du Milieu), on peut enrichir cette requête en identifiant :
Concepts liés à "Gondor" → Minas Tirith, Osgiliath, Númenor (ancêtres du Gondor)
Personnages associés → Aragorn (roi du Gondor), Boromir et Faramir (capitaines du Gondor), Denethor (intendant du Gondor), Isildur et Anárion (fondateurs)
Relations indirectes → Un graphe comme celui de Wikidata ou une base spécialisée pourrait relier "Gondor" à "Númenor", ce qui permettrait d'inclure des personnages ayant un lien historique avec le royaume.
Requête enrichie après expansion :
"Personnages du Gondor, y compris les rois et intendants, les anciens Númenóréens ainsi que les habitants de Minas Tirith et Osgiliath."

Grâce à cette expansion, la recherche ne se limite pas aux personnages directement mentionnés comme étant du Gondor, mais inclut aussi ceux qui y ont régné, ceux qui ont influencé son histoire et ceux qui en sont originaires. Cela permet d'obtenir des résultats plus riches et pertinents.

5 https://medium.com/towards-data-science/langchains-parent-document-retriever-revisited-1fca8791f5a0

II) Les graphes de connaissance pour Augmenter les modèles génératifs

Si les LLMs disposent (pour des systèmes informatiques) de capacités inédites en terme de rédaction de textes fluides et qu'ils sont capables à partir d'instructions[6] de réaliser correctement une grande variété de tâches, la technologie n'est pas encore mature et ils ne peuvent pas encore être utilisés avec succès pour toutes les tâches génératives. On appellera « tâche générative » toute tâche attendant en output un texte rédigé. On peut considérer par exemple le résumé automatique abstractif, mais aussi la création automatique de contrats de location immobilière comme des tâches génératives. Si les LLM ont des performances très convaincantes sur la première, ils ne sont a priori pas capable seuls de créer des contrats juridiquement valides.

Les LLMs souffrent également de limitations majeures, notamment leur propension aux hallucinations (génération de contenu non factuel[7]) du à la limitation de leurs « connaissances » aux informations contenues dans leurs données d'entraînement. On parle de connaissances paramétriques. C'est pourquoi les LLMs n'ont qu'une connaissance très limitée des domaines de spécialité et ont besoin d'être connectés à des sources extérieures pour gérer des tâches relevant de ces domaines. On présentera comment les graphes de connaissance sont des sources potentielles pour ancrer la réponse des LLMs dans des faits et pour leur donner accès à la compréhension d'un vocabulaire particulier. Encore faut-il s'assurer que les LLMs peuvent comprendre et interagir avec le contenu de ces graphes.

1) Le graphe et les LLMs : des interaction mitigées

A priori, Llm et graphes ne sont pas exactement faits pour s'entendre. Les graphes sont un format de données structurées, alors que les LLMs sont entraînés sur des données non structurées. En faisant attention à en inclure en nombre dans les corpus d'entraînement, les créateurs des modèles se sont assurés que les LLMs puissent traiter correctement certains formats de données structurées notamment les json, et dans une moindre mesure le xml. Le même genre de travail est en cours pour les données tabulaires (en passant par du markdown, html…) mais il n'y a pas eu de consensus entre les créateurs de modèles sur la manière de procéder si bien que la stratégie et les résultats sont très disparates d'un modèle à l'autre. En tout cas aucun des LLMs les plus connus ne contient de graphes de façon systématique dans son corpus d'entraînement. Cela amène un certain nombre de limitations et de questions.

* Les graphes de connaissances peuvent-ils servir de source utiles pour les LLMs ? Vont-ils comprendre l'information stockée dans le formalisme spécifique du triplet ?

* Si on ne donne pas seulement quelques triplets mais tout une portion du graphe, les LLMs vont-ils être capables de comprendre les relations entre les entités ? Comment représenter les relations pour s'en assurer ?

* Cela peut-il passer à l'échelle sur de gros graphes ?

6 https://arxiv.org/abs/2005.14165 papier à connaître sur la question : LLM are few shot learners
7 https://arxiv.org/abs/2311.05232 survey de référence sur la question des hallucinaitons

Ces questions ont donné lieu à une série de travaux tentant d'évaluer la capacité des LLMs à raisonner et effectuer des opérations sur les graphes. Des premiers travaux ont tenté de mettre à jour les capacités de raisonnement des LLM sur les graphes et notamment si les LLMs étaient capables de répondre à des questions simples comme le nombre de voisins d'un nœud, la découverte de chemins… c'est à dire voir si le LLM comprend la donnée structurée. Dans un second temps, les tests peuvent porter sur la capacité des LLMs à effectuer des opérations de déduction et répondre à des questions plus complexes.

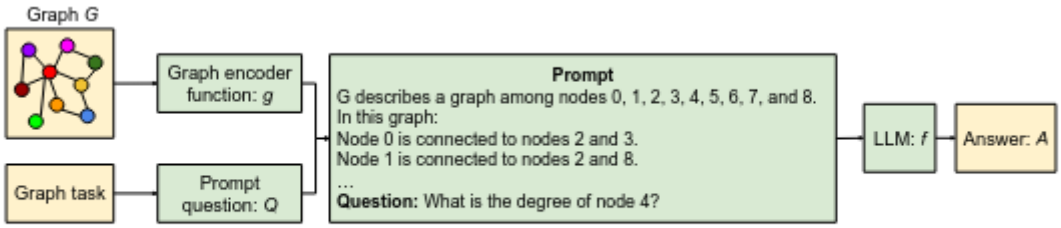


Figure 1: Overview of our framework for reasoning with graphs using LLMs.

L'article *"Talk like a Graph: Encoding Graphs for Large Language Models"*[8] explore comment représenter efficacement des données sous forme de graphes pour les traiter avec des modèles de langage à grande échelle (LLMs). L'objectif est de comprendre comment différents facteurs influencent la capacité des LLMs à raisonner sur des structures de graphe et à effectuer des tâches spécifiques. Certaines des conclusions importantes sont que :

* la manière dont les graphes sont transformés en texte a un impact direct sur la performance des modèles de langage.

* les LLMs sont évalués sur plusieurs types de tâches complexes, notamment la classification de nœuds (prédire la catégorie d'un nœud donné en fonction de son voisinage), la prédiction de liens (déterminer si une connexion existe entre deux nœuds) et le raisonnement global sur la structure du graphe (analyser des motifs complexes). Les performances varient selon la complexité des tâches et la manière dont l'information est fournie.

* La topologie du graphe (la taille, la densité et la complexité du graphe) influence la capacité du LLM à raisonner efficacement. Certains types de graphes sont plus difficiles à encoder en langage naturel sans perdre d'information critique.

D 'où l'importance de choisir un bon encodage textuel des graphes pour améliorer les performances des modèles de langage sur des tâches impliquant des relations complexes. Si les LLMs ne soient pas conçus spécifiquement pour le raisonnement sur graphes, une bonne transformation des données peut considérablement améliorer leur efficacité.

Une seconde série de travaux porte sur les différentes techniques que l'on pourrait utiliser pour rendre les LLMs mieux capables de comprendre les graphes. Là où la plupart des travaux sur la question se contentent de fournir des triplets de graphe dans la fenêtre de contexte d'un LLM pour répondre à des questions, certaines équipes[9] ont fine tuné le LLM sur des triplets de graphes

8 https://openreview.net/pdf?id=IuXR1CCrSi Talk like a graph de Google Research
9 https://arxiv.org/pdf/2310.01061

pour les rendre plus à même de comprendre la structure de l'information. L'inconvénient est qu'il faut disposer d'un corpus spécialisé au cas d'usage et le refaire sur chaque LLM utilisé.

D'un point de vue technique, il est possible de faire interagir un LLM avec les GNN spécifiquement conçus pour traiter les graphes. Une étude assez complète[10] envisage les modes d'interaction possibles. Les auteurs identifient trois principaux contextes d'application qui dépendent de la nature du graphe :

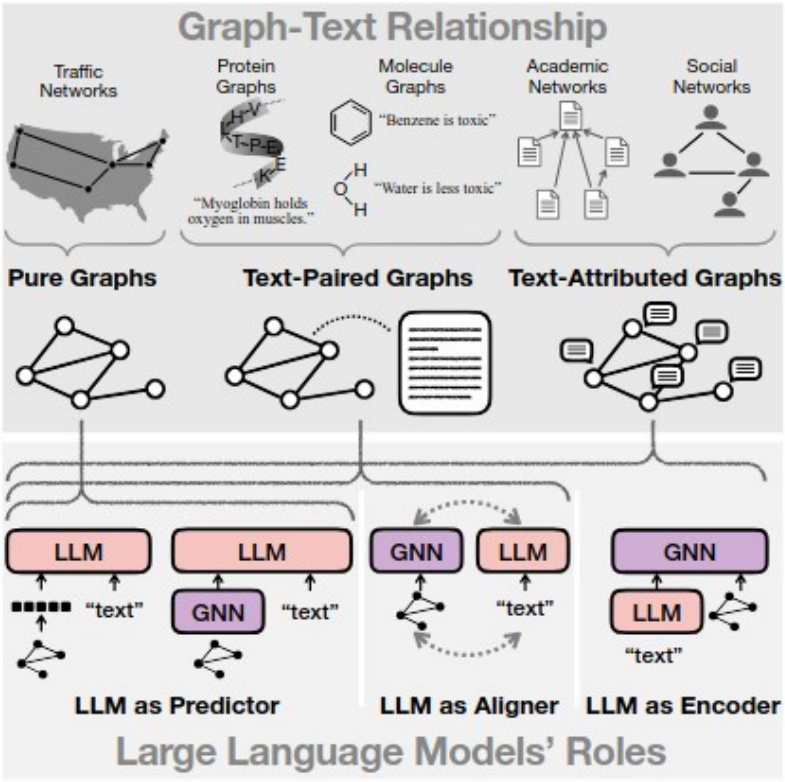
Graphes purs : où les données sont principalement structurées sous forme de graphes sans attributs textuels significatifs.

Graphes avec attributs textuels : où les nœuds ou les arêtes des graphes sont enrichis d'informations textuelles, comme des descriptions ou des étiquettes.

Graphes jumelés avec du texte : où les graphes sont associés à des textes complets, tels que des articles scientifiques liés à des réseaux de citations.

Pour chaque scénario différentes approches d'intégration des LLMs sont possibles:

- **LLM en tant que prédicteur** : les LLMs sont utilisés pour effectuer des prédictions directement à partir des structures de graphes.
- **LLM en tant qu'encodeur** : les LLMs servent à encoder les informations textuelles associées aux graphes, facilitant ainsi une meilleure représentation des données.
- **LLM en tant qu'aligneur** : les LLMs alignent les représentations textuelles et structurées pour améliorer la cohérence et l'efficacité des modèles.



Ces travaux bien qu'encore exploratoires font montre d'une grande créativité pour traiter un problème technique qui n'est vraiment pas trivial.

Si cette recherche n'aura vraisemblablement pas d'écho direct sur la manière dont nos LLMs sont entraînés, elles pourraient potentiellement apporter des avancées majeures dans les applications où des LLMs sont appariés à des graphes de connaissances.

10 https://arxiv.org/pdf/2312.02783

2) le RAG classique et ses limites

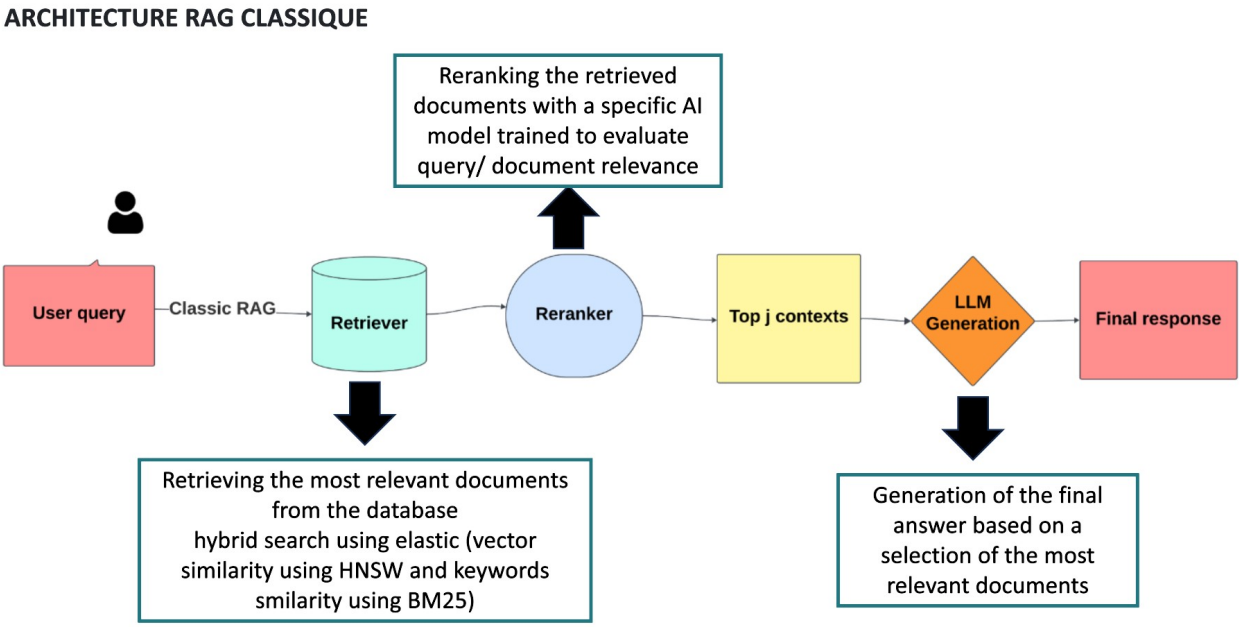
Étant donné que les LLMs ne sont pas conçus nativement pour comprendre des graphes, la plupart des stratégies explorées pour palier les limitations des LLMs et les connecter à des sources d'information issues de données textuelles passent donc par l'usage de données non structurées. On étudiera comment les LLMs ont pu être complétés par des données structurées pour ensuite voir comment transposer cette approche aux données structurées que sont les graphes. On présentera notamment l'approche RAG qui fait fureur dans l'industrie.

Avant l'avènement des LLMs, une approche prometteuse avait été proposée par Lewis et al.[11] pour connecter des bases de données spécialisées avec des modèles génératifs, à l'époque des modèles de type BART dont le décodeur était spécialisé dans la génération de texte. Cette nouvelle architecture s'est depuis imposée comme une alternative prometteuse au fine-tuning dont le coût serait prohibitif sur des modèles de grande taille. Cette architecture combine une phase de recherche d'information (retrieve) visant à sélectionner dans la base de données les k documents les plus susceptibles de répondre à la question, avec une phase de génération lors de laquelle les documents sont fournis comme entrée (input) au modèle génératif en même temps que la question. Pour cela, les documents ont été indexés dans une base de données vectorielle au moyen d'un modèle de type encodeur qui est également utilisé pour construire une représentation de la question donnée en entrée du modèle. Une simple mesure de similarité entre l'embedding de la question et celui de la réponse permet de classer les documents de la base de données pour ne sélectionner que les meilleurs. Cette première version du RAG est le premier modèle hybride qui combine des connaissances paramétriques (apprises lors de l'entraînement) et des connaissances non paramétriques (issues de la base de données 4) avec comme finalité la génération d'une réponse à une question. Au départ, le RAG n'était pas une approche dissociée du fine-tuning car dans l'article original, le système était entraîné de manière end-to-end : le modèle encodant la question de l'utilisateur et les documents de la base, ainsi que le modèle assurant la génération de la réponse finale étaient entraînés en même temps, précisément en fine-tunant des modèles génériques.

Ce n'est qu'après que les LLMs ont remplacé les modèles de type BART dans les tâches génératives que le RAG a été considéré comme une architecture (faite de briques et de modèles différents et pouvant être remplacés) constituant une alternative au fine-tuning, et plus un modèle à part entière. Si cette architecture fait l'objet de publications régulières, c'est surtout dans l'industrie qu'elle s'est imposée par sa simplicité, car elle répondait au besoin des entreprises de connecter les modèles de langue à leurs bases de données privées afin de construire par exemple des chatbots ou des outils d'analyse ayant accès à des données spécifiques. De nombreuses librairies telles que LangChain, LlamaIndex ou encore Haystack ont fleuri et permettent en moins de 100 lignes de code de construire un système de questions-réponses, c'est à dire l'indexation des documents, l'extraction des documents les plus pertinents et l'intégration de ces derniers dans la génération finale d'un LLM fournissant une réponse à la question donnée en entrée du système. Et les blogs de data science de s'extasier de la simplicité et de l'efficacité de cette approche comme si la question

11 https://arxiv.org/abs/2005.11401 Bien que ce soit leur article de 2020 qui ait été le plus largement repris quand l'intuition a été popularisée, plusieurs travaux contemporains voire antérieurs creusaient déjà cette idée de combiner rechercher d'information et génération de texte.

de la connexion entre des bases de données externes et les LLMs était résolue… Les résultats triomphaux présentés dans la page de présentation des librairies cachent une réalité moins idyllique : si une implémentation naïve de l'architecture RAG sur une base de données de taille raisonnable (quelques centaines de documents) et diverse ou sur des benchmarks de Question Answering fournit de bons résultats, son application à large échelle et sur des données plus spécialisées et homogènes est difficile.



L'architecture classique du Retrieval-Augmented Generation (RAG) repose sur une combinaison de recherche d'informations et de génération de texte afin de produire des réponses plus précises et contextuelles. Ce schéma détaille les différentes étapes du processus.

Tout commence par la soumission d'une requête utilisateur.

* Cette requête est ensuite traitée par un **retriever**, dont le rôle est d'identifier et de récupérer les documents les plus pertinents à partir d'une base de données. Pour ce faire, il utilise une recherche hybride, combinant deux méthodes complémentaires : La recherche vectorielle avec HNSW (Hierarchical Navigable Small World), qui permet d'évaluer la similarité sémantique entre la requête et les documents. L'algorithme BM25, qui repose sur une approche plus classique basée sur la correspondance de mots-clés.

* Une fois les documents récupérés, ils sont soumis[12] à un **reranker**, un modèle d'intelligence artificielle chargé de réévaluer et de reclasser les documents en fonction de leur pertinence par rapport à la requête. Cette étape permet d'affiner la sélection et d'améliorer la qualité des résultats.

* Les meilleurs documents (ou top j contexts) issus de cette sélection sont ensuite transmis à la **phase de génération de texte**. Un modèle de langage (LLM) est utilisé pour produire la réponse

12 Ce qui n'était pas présent dans l'architecture originale du papier de 2020 mais a fini par s'imposer dans l'industrie et être adopté comme la norme par Amazon, Elastic, MongoDB en 2024. Les travaux de recherche démontraient fermement l'amélioration notable de précision apportée depuis au moins 2022…

finale, en s’appuyant sur les documents les plus pertinents. Cette approche permet d’obtenir des réponses mieux informées et plus contextualisées, dépassant ainsi les limites des modèles de langage génératifs traditionnels qui reposent uniquement sur leurs connaissances pré-entraînées.

Si cette approche a permis de connecter avec succès des LLMs à des bases de données extérieures, et des données provenant de domaines de spécialité, elle souffre de quelques limitations fondamentales liées à ses composants :
* l’architecture actuelle est trop sensible à la formulation précise de la question de l’utilisateur et la recherche de documents ne renvoie dès lors pas forcément les meilleurs passages pour y répondre. Cela peut être mitigé en s’inspirant de techniques d’Information Retrieval déjà mises en place dans les moteurs de recherche[13].
* puisqu’un nombre arbitraire de k documents sont sélectionnés, sans pénaliser de plus la redondance d’information entre les documents, la réponse à une question pourrait ne contenir qu’une partie des éléments de réponse contenus dans la base de données (question demandant de constituer une liste par exemple).
* Les modèles d’embedding utilisés ont été conçus à partir de texte rédigé de qualité et ne sont pas habitués à traiter d’autres types de contenus (données tabulaires, données bruitées de mauvaise qualité comme des messages de chat ou des posts de réseaux sociaux). Il faut que ces documents soient aussi pris en compte si les informations contenues sont pertinentes.
* le système actuel emploie des documents comme source de la réponse, mais parfois la réponse à une question n’est pas contenue intégralement dans un document de la base de données mais peut être obtenue par combinaison de plusieurs informations ou plusieurs documents qui ne seront pas forcément ceux sélectionnés par le système.

Les systèmes RAG classiques ont du mal à identifier les connexions entre différentes entités, à moins que ces concepts n'apparaissent explicitement dans le même document. Ces cas évidents ne couvrent pas toute la diversité des requêtes des utilisateurs, ce qui entraîne une faible performance en rappel.

Prenons, par exemple, la requête suivante :
"Quels personnages ont possédé l’Anneau unique avant Frodon ?"

Un RAG classique ne récupérerait que des documents contenant des réponses explicites, comme :
"Bilbon a trouvé l’Anneau dans la grotte de Gollum et l’a transmis à Frodon."

Cependant, si l’information est dispersée dans plusieurs documents :
"L’Anneau unique fut forgé par Sauron..."
"Isildur arracha l’Anneau à Sauron avant de le perdre dans l’Anduin..."
"Des siècles plus tard, Sméagol (plus tard connu sous le nom de Gollum) trouva l’Anneau..."
Un système RAG classique pourrait échouer à reconstituer l’ensemble de la chaîne de possession.
En revanche, un graphe de connaissances relierait ces personnages (Sauron → Isildur → Gollum → Bilbon → Frodon) et permettrait de répondre correctement à la requête en identifiant toutes les connexions pertinentes, même si elles ne figurent pas explicitement dans un seul document.
D’où la nécessité de revenir sur nos graphes !

13 https://caid-conference.eu/wp-content/uploads/2024/11/CAID2024_paper_29.pdf

3) Le Graph RAG et les différentes approches proposées

Finissant de constater les limitations du RAG « classique » sur documents, les chercheurs ont rapidement pensé à se servir de graphes de connaissances à la place d'index de documents pour fournir aux LLMs des informations contextuelles.

RAG sur des documents	RAG sur un graphe
Facile à mettre en œuvre	Simple sur le principe / Implémentation non triviale Comment et quoi récupérer ? Comment le fournir au LLM?
Possibilité de ne fournir qu'un nombre fini de contextes (limite technique de la fenêtre de contextes)	Focus sur un triplet (et éventuellement une phrase) Possibilité de fournir une plus grande fraction de la base de connaissances dans le prompt.
Incapacité de faire la connexion entre des informations présentes dans plus de k documents	Possibilité (cf propriétés mathématiques des graphes) de déterminer facilement le lien (ou l'absence de lien) entre des entités nommées et d'utiliser les informations présentes dans tous les documents.

C'est ainsi que le terme de Graph RAG est apparu. Si l'intuition derrière le Graph RAG semble intuitive (on requête les informations utiles dans le graphe pour répondre à la question de l'utilisateur), dans la pratique, l'implémentation d'un tel système est beaucoup moins claire, si bien que le terme de « Graph RAG » a fini par désigner des techniques et des approches qui n'avaient rien à voir entres elles.

Si l'on essaie d'avoir une vue analytique et de prendre du recul sur la masse de travaux publiés :

1) Extraction de nœuds et de relations

Cette approche consiste à intégrer les composants d'un graphe de connaissances, c'est-à-dire ses sommets (nœuds) et ses arêtes (relations), en utilisant des techniques d'intégration pertinentes alignées avec la méthode d'intégration utilisée pour la requête. Ces embeddings sont ensuite récupérés en fonction de leur similarité vectorielle. Par exemple, dans l'étude "Graph Reasoning for Question Answering with Triplet Retrieval" (Li et al., 2023)[14], les auteurs proposent de linéariser les triplets du graphe en phrases et de les intégrer pour récupérer les triplets les plus pertinents. Cependant, cette méthode ressemble étroitement au RAG classique, où les triplets du graphe agissent comme des "fragments". Elle ne parvient pas à exploiter pleinement les propriétés mathématiques et structurelles uniques des graphes de connaissances, telles que les chemins de relation et le parcours de graphe.

2) Regroupement de graphes et résumé de clusters

Cette technique implique de regrouper des nœuds similaires en clusters et de sélectionner les clusters les plus pertinents pour répondre à la requête. En résumant les clusters, le système réduit la complexité du graphe avant d'interagir avec le LLM. Bien qu'innovante, cette méthode est coûteuse en termes de calcul, en particulier pour les graphes de grande taille avec des données de haute dimension. Elle a surtout été connue via un article de Microsoft Research[15] mais je ne connais personne qui fasse cela dans la vraie vie.

14 https://arxiv.org/abs/2305.18742?utm_source=chatgpt.com
15 https://arxiv.org/html/2404.16130v2?utm_source=chatgpt.com

3) Transformation de la requête en requête de graphe

Inspirée par les techniques de text-to-SQL, cette approche convertit la requête en langage naturel de l'utilisateur en une requête de base de données de graphes (par exemple, en utilisant Cypher pour Neo4j). La requête de graphe est ensuite exécutée pour extraire le sous-graphe le plus pertinent que le LLM doit traiter. Cette approche ne fonctionne malheureusement que pour les requêtes qui peuvent être efficacement traduites en requêtes de type base de données. Elle nécessite le stockage des données dans une base de données de graphes capable d'exécuter de telles requêtes, ce qui impliquerait de maintenir deux infrastructures de stockage distinctes pour les architectures RAG hybrides (une pour les documents et une pour les graphes).

Bien que prometteuses, ces stratégies présentent des défis significatifs, tels que le coût computationnel, une évolutivité limitée et une complexité de l'infrastructure. Or on souhaiterait tirer parti des puissantes propriétés des graphes de connaissances sans doubler l'infrastructure de stockage ou devoir réaliser des opérations trop coûteuses. Il convient donc de penser à des stratégies concrètes (en d'autre termes une implémentation) du Graph RAG qui soit rapide et efficace.

III) Présentation d'une implémentation de Graph RAG par génération à la volée de sous graphes de connaissance

On tentera de montrer que l'implémentation d'un système tel que le Graph RAG, dont la description théorique est pourtant fort simple, nécessite un vaste panel de compétences en NLP (gestion d'embeddings), en ingénierie des connaissances (construction de graphe de connaissances, recherche des informations), en algorithmique (optimisation et travail sur les graphes) et une bonne maîtrise des contraintes techniques liées à l'utilisation de LLM. On se consacrera sur une implémentation particulière du Graph RAG qui se veut peu coûteuse et rapide. La description plus détaillée (mais en anglais) de cette approche est disponible sur une page du site d'Elasticsearch[16].

En effet, procéder à des recherches sur des graphes de grande taille (millions de nœuds) peut s'avérer coûteux car les opérations sur les graphes sont en général plus que linéaire en fonction du nombre de nœuds. L'idée de cette technique est de ne jamais requêter le graphe entier mais de générer à partir d'une structure de stockage des triplets adaptée le sous graphe pertinent par rapport à la question de l'utilisateur. Les opérations sur le graphe ont lieu sur ce graphe réduit (quelques centaines de nœud au plus) et sont donc très peu coûteuses. La réponse à la question se fait en quatre étapes :

1) extraction des entités nommées et des concepts dans la question de l'utilisateur

2) à partir de ces résultats, génération d'un sous graphe de connaissance connectant ces entités.

3) simplification du graphe (pruning) pour ne conserver que les triplets pertinents.

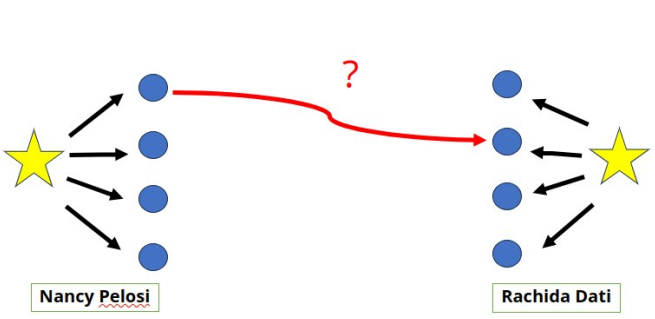
4) linéarisation des triplets pour les fournir comme contexte au LLM. Génération de la réponse finale.

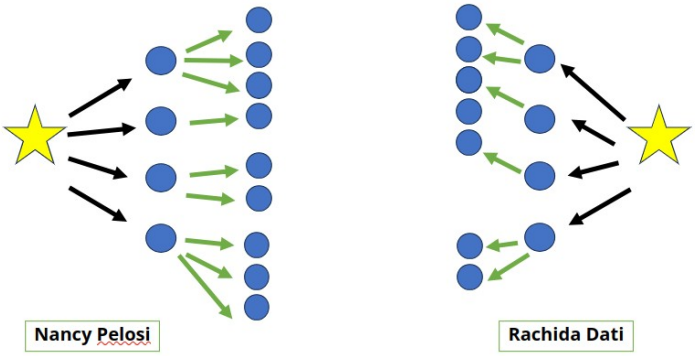
16 https://www.elastic.co/search-labs/blog/rag-graph-traversal

1) construire un sous graphe de connaissance à la volée

Pour éviter de lancer des requêtes coûteuses sur l'ensemble du graphe, l'idée est de générer un sous graphe à partir de la question de l'utilisateur. Pour cela on extrait les concepts et entités de la question et on va essayer de construire un graphe dans lequel ces entités sont connectées.
Les relations sont stockées dans un index NoSQL sous forme de triplets (from, relation, to) ce qui permet de faire facilement une recherche filtrée. Nous exploitons cette capacité pour étendre progressivement une recherche à partir des entités de la requête en suivant la procédure suivante :

Vérification de la connexion entre deux entités
*Nous vérifions d'abord s'il existe une relation directe entre les deux entités.
/Si ce n'est pas le cas, à l'aide d'une requête filtrée, nous récupérons la liste des nœuds connectés à chacune des deux entités. On vérifie si l'index des relations contient au moins une connexion entre un élément lié à la première entité et un élément lié à la seconde.
* Si une connexion est trouvée, nous arrêtons l'expansion du graphe.
* Sinon, nous répétons le processus en examinant tous les voisins directs des nœuds connectés aux deux entités.
Nous limitons le nombre d'itérations à trois, car deux entités connectées par plus de six sauts sont généralement faiblement reliées.





La taille du sous-graphe ainsi généré est imprévisible et dépend entièrement de la requête et du contenu du dataset. La seule contrainte que nous imposons lors de cette génération dynamique est de ne pas collecter plus de 100 voisins par nœud. Sune connexion directe existe entre les deux entités, le graphe contiendra au maximum :
2 nœuds principaux (Nancy Pelosi et Rachida Dati)
2 × 100 = 202 nœuds (leurs voisins immédiats)
201 arêtes reliant ces nœuds.
Pire cas : en raison de la limite par défaut de 10 000 résultats par requête, chaque phase d'expansion peut ajouter au maximum 10 000 nouveaux nœuds par côté. Ainsi, pour une requête impliquant deux entités, le graphe pourrait atteindre une taille maximale de :

3 (nombre d’itérations) × 2 (expansion des deux côtés) × 10 000 = 60 002 nœuds.
En réalité, de telles tailles ne sont jamais atteintes. Cela s’explique par la topologie des graphes de connaissances, qui comportent généralement :
* Quelques "hubs" (nœuds à forte connectivité).
* Beaucoup de nœuds à faible connectivité, avec seulement un voisin unique dans la base de données (et qui n’étendent donc pas la recherche).
En moyenne, un nœud possède entre 10 et 20 connexions.

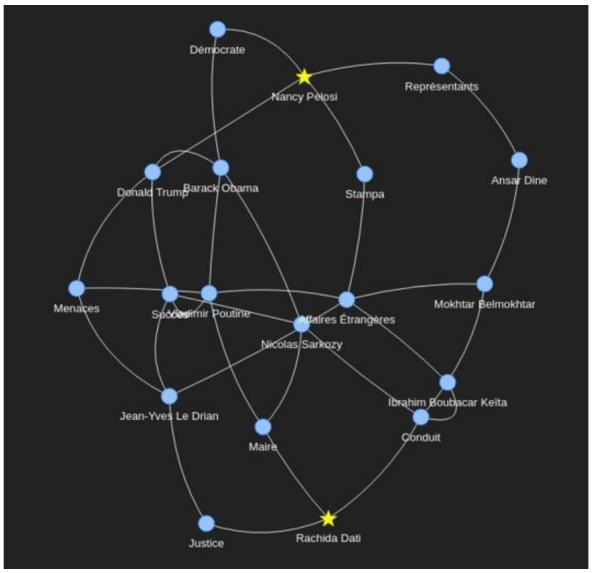
Même si des entités fortement connectées apparaissent dans le processus, la limitation à 100 relations par entité garantit que le sous-graphe généré dépasse rarement 1 000 nœuds.

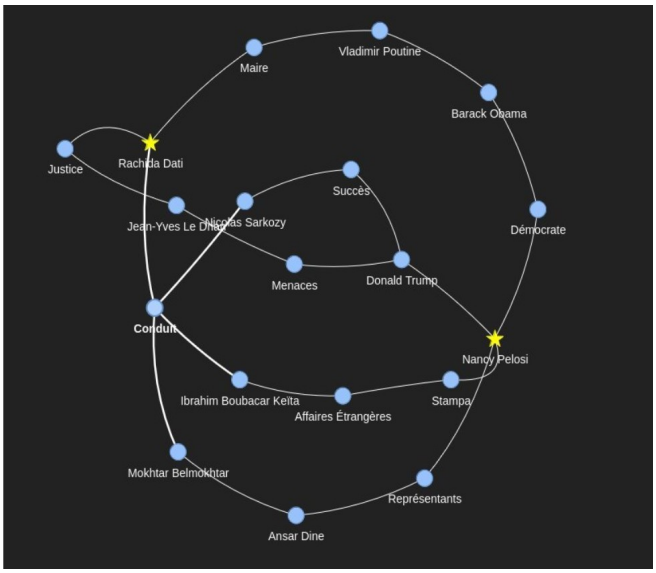
2) simplifier le sous graphe

Ce sous-graphe (quelques centaines de relations) est suffisamment petit pour permettre des opérations sur le graphe à un coût très faible, mais reste trop volumineux pour être directement exploité par un LLM pour la génération de réponses. Il est donc nécessaire d’appliquer une heuristique de simplification :

Première étape : Sélection des relations sur le chemin le plus court
Nous ne conservons que les relations appartenant aux chemins les plus courts reliant les entités d’intérêt. Cela permet d’éviter les triplets bruités et de ne conserver que les liens les plus pertinents.

Deuxième étape : Élagage du graphe
Si l’ensemble des relations restantes est encore trop volumineux, nous appliquons un algorithme de réduction du graphe. Cet algorithme :
* Réduit le nombre de relations, tout en minimisant la suppression des chemins les plus courts.
* Préserve la diversité des entités, en s’assurant que les entités apparaissant sur ces chemins restent bien représentées.
Grâce à cette approche, nous obtenons un sous-graphe optimisé, plus compact et exploitable par un LLM pour la génération de réponses.

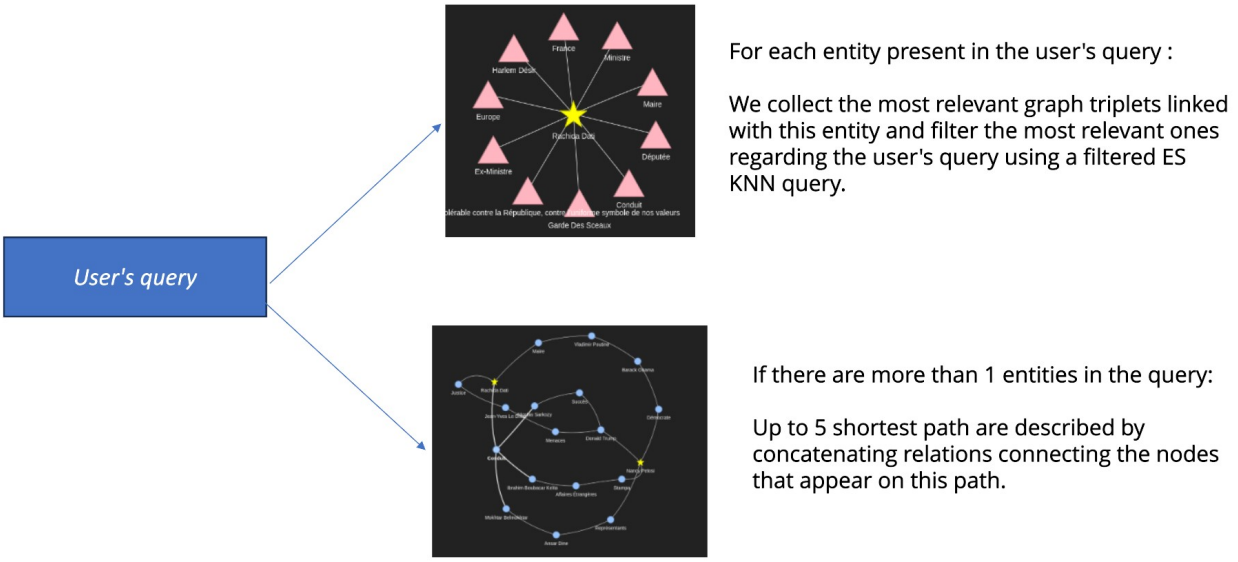




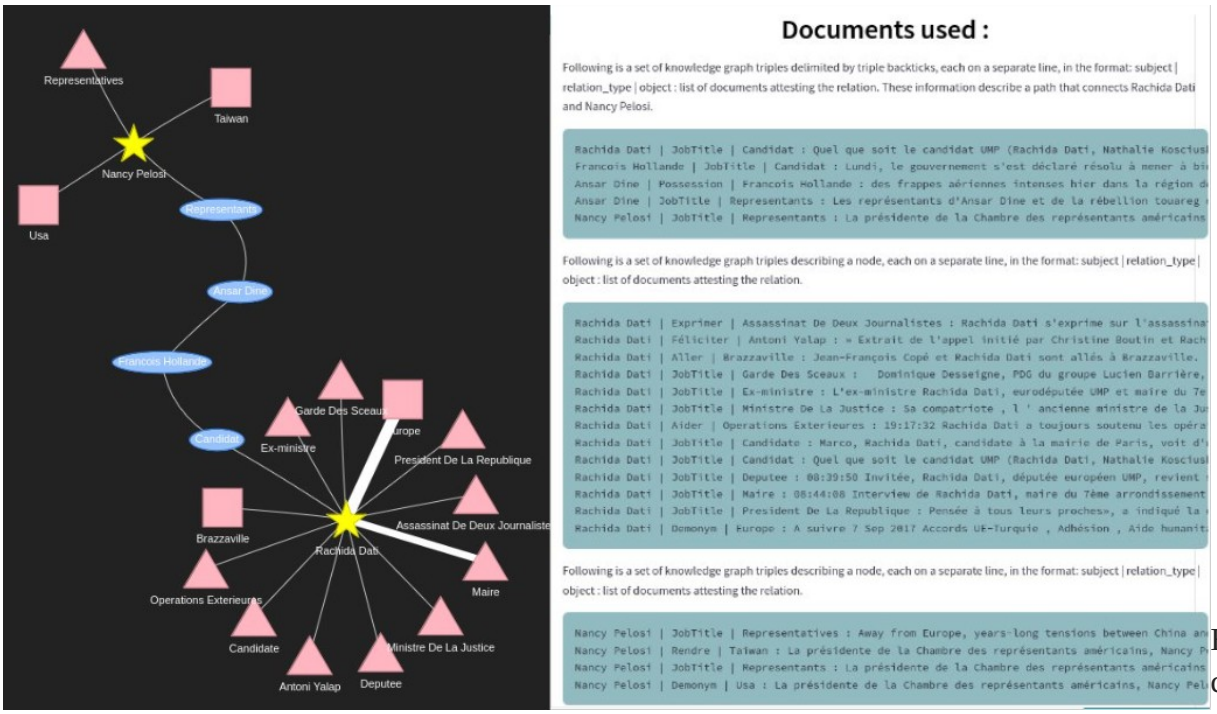
3) exploiter la réponse du sous graphe

Pour répondre à la requête de l’utilisateur, nous sélectionnons la section pertinente du graphe, mais il est nécessaire de la convertir dans un format exploitable par un LLM pour générer la réponse finale. Nous avons choisi de linéariser le graphe sous forme textuelle en générant deux types de "pseudo-documents" à partir du graphe.

Deux approches pour créer des pseudo-documents à partir du graphe :
* on crée un document par entité principale (se trouvant dans la question)
* on crée un document par chemin le plus court



Ces pseudo-documents sont construits en concaténant les triplets du Knowledge Graph (KG) dans un format lisible. Pour améliorer la compréhension et fournir des preuves à l’appui, nous intégrons également des phrases extraites des documents de la base de données qui illustrent chaque triplet.

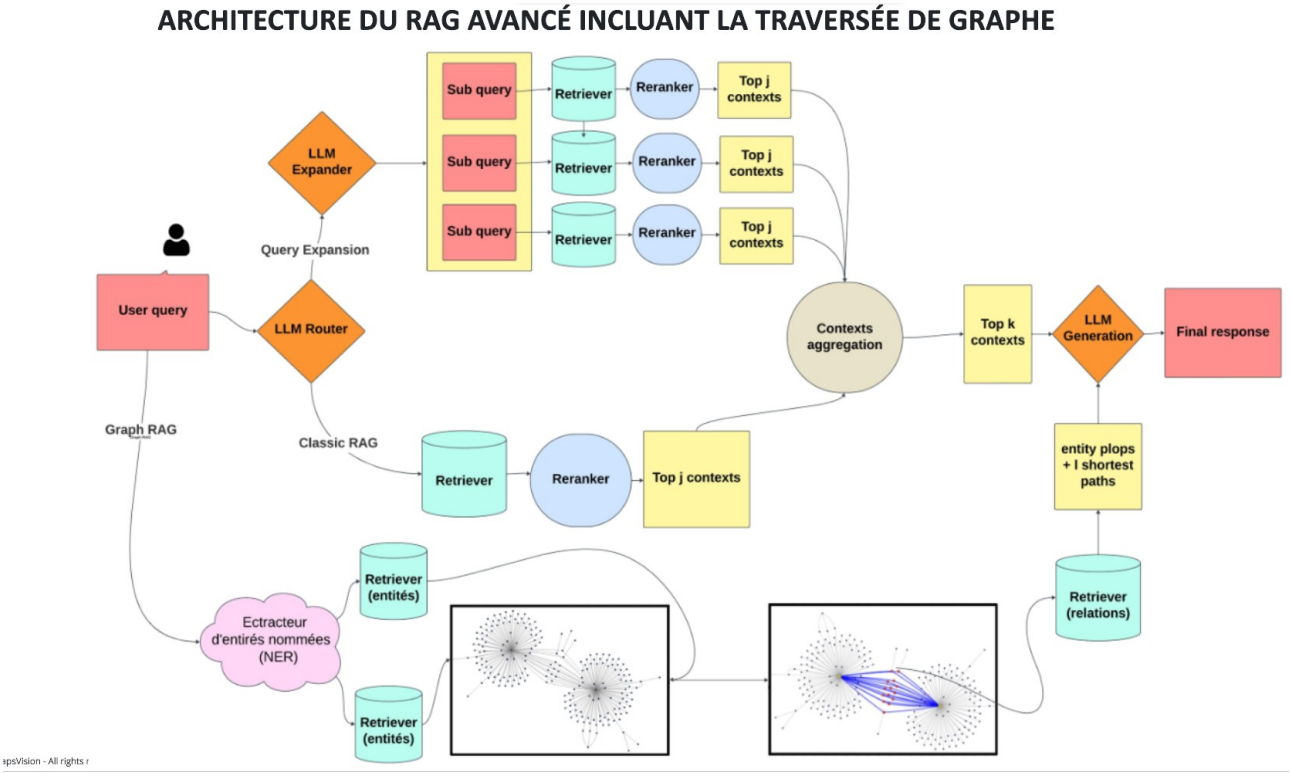


Enfin, ces documents sont transmis au LLM, soit en remplacement, soit en complément des documents récupérés par un retriever RAG classique, améliorant ainsi la qualité et la fiabilité des réponses générées.

En conclusion, le Graph RAG permet d'obtenir de meilleurs résultats que le RAG classique pour des questions qui attendent de rétablir le lien entre plusieurs entités ou d'être exhaustif sur une question (les triplets étant plus courts on est moins impacté par la taille réduite de la fenêtre de contexte des LLM).

Néanmoins le Graph RAG a ses limitations : * la donnée structurée (triplet) propose moins d'informations riches, de sémantisme, que des phrases complètes. On peut limiter cet aspect en fournissant comme on l'a fait en plus du triplet la phrase qui a été à l'origine de l'ajout du triplet.
* Avec seulement un triplet comme information, le LLM peut aller trop loin dans l'inférence du lien entre les entités et donc halluciner.
* Cette méthode est sensible à la formulation exacte des termes et des concepts (recherche exacte de l'élément comme nœud...)
* Une relation entre deux objets marque les relations qui apparaissaient dans le même contexte. Ces dernières n'ont pas plus de valeur.

C’est pourquoi le Graph RAG ne peut pas seul se substituer au RAG dans tous les cas d’usage et pour toutes les questions. Il est dès lors préférable de mettre en place un approche hybride qui combine recherche documentaire et recherche dans le graphe de connaissances pour informer la réponse finale du LLM. Si l’on prend un peu de recul, un telle architecture hybride est complexe.



Plus que jamais, l’ingénierie des connaissances devient aussi une ingénierie logicielle qui va au-delà du NLP simple.

TODO petite conclusion