ELO-GRAG@EvalLLM 2026 : vers un système HybridRAG combinant graphes de connaissances et texte, évalué sur un corpus de défense en français

Thibault Ehrhart[1] Carmelle Meli Songuon[1, 2] Julien Plu[3] Raphael Troncy[1]
Yoan Chabot[2] Edouard Trouillez[3] Oscar Moreno Escobar[3]
(1) EURECOM, Sophia Antipolis, France
(2) Orange Research, Belfort, France
(3) Lettria, Paris, France
{carmelle.melisonguon, yoan.chabot}@orange.com,
{thibault.ehrhart, raphael.troncy}@eurecom.fr,
{julien, edouard, oscar}@lettria.com

RÉSUMÉ

Nous présentons ELO-GRAG, un système soumis au challenge RAG de EvalLLM 2026. Le corpus officiel est composé de 375 documents totalisant 10 741 pages. Nous le convertissons tout d'abord en Markdown annoté afin d'extraire le texte tout en préservant la structure documentaire et la provenance avec comme unité la page. Pour la tâche 1, nous comparons un RAG combinant recherche dense et BM25, le système Microsoft GraphRAG, ainsi que deux variantes hybrides exploitant un graphe de connaissances RDF extrait automatiquement. Pour la tâche 2, nous formulons l'attribution des sources comme un jugement contrôlé par un LLM, avec ou sans enrichissement par triplets verbalisés. Les premiers résultats montrent que le RAG vectoriel constitue notre meilleur système de recherche de preuves, tandis que le meilleur compromis pour l'attribution de sources est obtenu par un juge Gemma utilisant uniquement le texte des pages candidates.

ABSTRACT

ELO-GRAG

We describe ELO-GRAG, our system submitted to the EvalLLM 2026 RAG Challenge. The official corpus contains 375 documents, for a total of 10 741 pages. We convert this corpus into annotated Markdown in order to extract the textual content while preserving the document structure, and page-level provenance. For Task 1, we compare a RAG system combining dense and lexical retrieval, Microsoft GraphRAG, and two hybrid variants that use an automatically extracted RDF knowledge graph. For Task 2, we formulate source attribution as a controlled LLM-as-a-judge task, with or without additional verbalized triples from the knowledge graph. Preliminary results show that our vector RAG system is the strongest evidence retrieval method, while the best source attribution trade-off is obtained by a Gemma judge using only the textual content of candidate pages.

MOTS-CLÉS : GraphRAG, génération augmentée par récupération, graphes de connaissances, attribution de sources, EvalLLM.

KEYWORDS: GraphRAG, retrieval-augmented generation, knowledge graphs, source attribution, EvalLLM.

1 Introduction

La génération augmentée par la récupération (Retrieval-Augmented Generation, RAG) est une approche efficace utilisée pour améliorer les capacités des modèles de langue. En les équipant d'une source documentaire externe, les systèmes RAG peuvent produire des réponses plus précises, plus à jour, ou mieux adaptées à un domaine (Lewis *et al.*, 2020). L'approche GraphRAG propose d'exploiter la structure des graphes pour améliorer les performances des systèmes dans des scénarios qui nécessitent de capturer des relations ou des dépendances complexes que le RAG traditionnel pourrait manquer (Edge *et al.*, 2025; Peng *et al.*, 2025; Meli Songuon *et al.*, 2026). Cependant, l'évaluation de ces systèmes pose encore des difficultés. Les benchmarks existants ne reflètent pas toujours des cas d'usage réels où les sources à disposition sont bruitées, hétérogènes et impliquent des requêtes utilisateur complexes (Wang *et al.*, 2026). De plus, la majorité de ces benchmarks est centrée sur la langue anglaise, ce qui limite l'évaluation des systèmes RAG dans d'autres langues, notamment le français.

Dans ce contexte, l'atelier EvalLLM 2026, organisé dans le cadre de la conférence CORIA-TALN 2026, a proposé un challenge autour du RAG. L'objectif est d'évaluer des systèmes RAG sur un ensemble de documents en français couvrant des cas d'usage spécifiques. Le challenge comprend deux tâches. Une première qui vise à évaluer les capacités de recherche documentaire (*retrieval*) et de génération de la réponse finale, et une seconde qui porte sur l'évaluation de la capacité des systèmes à attribuer correctement des sources à une réponse donnée.

Dans le cadre de ce challenge, nous avons proposé quatre approches pour la tâche 1 : un RAG vectoriel combinant recherche dense et recherche lexicale, une adaptation de Microsoft GraphRAG, et deux approches hybrides qui exploitent des graphes de connaissances RDF extraits automatiquement. Pour la tâche 2, nous comparons plusieurs juges LLM chargés d'attribuer les sources, avec ou sans enrichissement par triplets verbalisés. Le code du système est disponible à `https://github.com/D2KLab/ELO-GRAG`. La suite de cet article décrit le traitement documentaire, les différentes méthodes que nous avons testé, les résultats obtenus et notre analyse de ces résultats.

2 Traitement des documents

Le système ELO-GRAG prend en entrée le corpus officiel, soit 374 PDF et une image PNG convertie en PDF, pour un total de 375 documents et 10 741 pages. Comme la soumission impose des preuves *(document, page)*, la page physique reste l'unité de provenance. Le Markdown annoté conserve le texte, l'ordre de lecture, les numéros de page, les boîtes de mise en page et les frontières structurelles, tandis qu'un fichier `source.txt` garde le nom exact du PDF d'origine.

Parsing documentaire. La chaîne de parsing combine l'extraction de mots par `pdfplumber` (Singer-Vine & pdfplumber contributors, 2026) en conservant les informations de page et de typographie, l'analyse de layout par DocLayout-YOLOv10 (Zhao *et al.*, 2024) et la reconstruction de tableaux par Table Transformer (Smock *et al.*, 2022). L'alignement entre mots et régions produit une séquence de blocs typés et ordonnés, incluant notamment titres, paragraphes, listes, légendes, tableaux et éléments visuels. Chaque bloc est sérialisé en Markdown et précédé d'un commentaire HTML contenant `type`, `pages`, `bboxes` et `split_candidate`, les boîtes englobantes étant normalisées dans le repère de la page. Le champ `split_candidate` encode une

priorité de coupure calculée à partir de la structure du document. La valeur 0 marque les frontières les plus fortes, notamment le premier élément d'une page et le premier titre du document, tandis que les titres et paragraphes suivants reçoivent des priorités plus faibles selon leur position hiérarchique. La recherche vectorielle ignore ce signal car elle reste intra-page, mais l'extraction de graphe l'utilise pour préserver la structure du document lors du découpage.

Après parsing, le corpus représente environ 3,47 millions de mots et 23,2 millions de caractères. La longueur des documents varie fortement, avec une médiane de 8 pages par document et un maximum de 1 180 pages pour le document le plus long.

Reconstruction de la vue page-texte. La vue *page–texte* est reconstruite en parcourant le Markdown annoté comme un flux de blocs. Le texte nettoyé entre deux commentaires HTML est affecté aux pages listées dans `pages`, tandis que les commentaires sont retirés du contenu transmis aux modèles. La table obtenue, composée de `parsed_doc_id`, `doc_name`, `text` et du chemin source, sert de représentation commune au Vector RAG, à l'entrée page par page de Microsoft GraphRAG, aux runs de la tâche 2, et à l'outil de visualisation (Section 6).

Découpage structurel pour l'extraction de graphes. Pour les runs hybrides, nous découpons le Markdown annoté en respectant les frontières documentaires plutôt qu'avec une fenêtre glissante. Les commentaires sont supprimés du texte mais reportés dans des éléments `dp_elements`, avec des offsets de caractères permettant de rattacher chaque chunk à ses pages. La taille cible est de 1 500 caractères, avec une limite dure de 3 000 caractères. Les fragments trop courts sont fusionnés avec un voisin compatible. Une passe finale fusionne aussi les sections de moins de 200 caractères afin d'éviter d'envoyer au modèle des chunks trop courts et pauvres. Ce paramétrage produit 14 392 chunks structurels. Chaque chunk est décrit par un manifeste indiquant son PDF source, son identifiant de document parsé, son identifiant de chunk, les pages couvertes et des `page_spans`. Ces derniers permettent de reconstruire un chunk multi-pages en preuves au format officiel *document–page*.

Extraction de graphes et index de triplets. Les 14 392 chunks structurels sont soumis un par un, sans redécoupage supplémentaire, à l'outil Text2Graph développé par Lettria (Plu *et al.*, 2025; Lisena *et al.*, 2026) en utilisant le modèle `google/gemma-4-31B-it` (Google DeepMind, 2026), servi via vLLM (Kwon *et al.*, 2023). Plus précisémment, l'outil est utilisé en mode extraction d'information ouverte, i.e. sans ontologie pour guider l'extraction des triplets. Le format de sortie JSON se limite aux faits directement supportés par le texte du chunk, et produit un graphe vide lorsque cela est justifié. Les sorties sont validées syntaxiquement, puis les graphes valides sont sérialisés en RDF/Turtle avec `pyoxigraph`. Ils sont ensuite reparsés avec `rdflib` pour construire un index JSONL de triplets. Les URI sont normalisées en noms locaux, chaque fait est verbalisé sous la forme *sujet – prédicat – objet*, et chaque entrée conserve sa provenance, du chunk source jusqu'aux `page_spans`. L'index final contient 360 362 triplets.

3 Tâche 1 : méthodes testées

3.1 Vector RAG

Notre premier run pour la tâche 1 est un RAG vectoriel utilisé comme système de référence. Il sépare la recherche de preuves de la génération de la réponse : le *retriever* produit une liste de pages candidates au format du challenge, puis le modèle génératif reçoit uniquement le texte de ces pages.

Indexation. Chaque document est reconstruit page par page depuis le Markdown, puis les pages longues sont découpées en fenêtres de 900 mots avec un recouvrement de 120 mots. Aucun chunk ne traverse une frontière de page, ce qui permet de convertir directement chaque résultat en *(document, page)*. Les 10 985 chunks obtenus conservent leurs métadonnées de provenance et sont indexés dans ChromaDB avec `Qwen/Qwen3-Embedding-8B` (Zhang *et al.*, 2025), servi localement avec Text Embeddings Inference. Un index BM25 est construit sur les mêmes textes.

Recherche de preuves. Pour chaque question, nous récupérons les 120 meilleurs chunks denses et les 200 meilleurs chunks BM25, puis les fusionnons par Reciprocal Rank Fusion (RRF) (Cormack *et al.*, 2009) pondéré, avec $w_{\text{dense}} = 1{,}0$, $w_{\text{BM25}} = 0{,}5$ et $k = 60$. Les chunks sont agrégés au niveau des pages, dédupliqués, limités à 10 pages au total et 3 pages par document, en conservant au moins 2 pages lorsque des candidats existent. La sélection s'arrête quand le score descend sous 75 % du meilleur score ou quand la baisse relative dépasse 18 %. Après cette sélection, la page suivant une page retenue peut être ajoutée lorsqu'elle existe et que la limite globale de 10 pages n'est pas atteinte, afin de conserver le contexte immédiat d'un passage situé en fin de page.

Génération de la réponse. La réponse finale est générée avec `google/gemma-4-31B-it`, servi via vLLM, avec une température fixée à 0. Le prompt impose de répondre en français uniquement à partir des sources fournies et de retourner un objet JSON contenant la réponse. Pour chaque question, au plus 10 pages de preuves et 30 000 caractères de contexte sont transmis au modèle. La sortie soumise contient la réponse générée et la liste des pages retrouvées, sans exposer les citations dans le texte de réponse.

3.2 Microsoft GraphRAG

Notre deuxième run pour la tâche 1 utilise le système GraphRAG de Microsoft [1] qui a été proposé afin de résoudre les limites du RAG vectoriel face aux requêtes de synthèse (Query Focused Summarization tasks, QFS) (Edge *et al.*, 2025). En effet, le RAG vectoriel, qui découpe les documents en fragments indépendants, entraîne une perte de contexte global, ce qui le limite pour les requêtes nécessitant une compréhension globale de l'ensemble des données (ex : « Quels sont les thèmes principaux de ce document ? »). Ce système GraphRAG exploite un graphe de connaissances et des résumés hiérarchiques de communautés construits à l'aide d'un LLM. Lors de la phase d'indexation, un LLM est utilisé pour extraire des entités et des relations du corpus de textes. Le graphe obtenu est ensuite partitionné en une hiérarchie de communautés à l'aide de l'algorithme de Leiden (Traag *et al.*, 2019), puis un résumé est généré pour chacune des communautés détectées.

Pendant la phase de réponse, le système propose deux principaux modes de requête :

- **Le mode de recherche local (*Local Search*) :** Il combine les données pertinentes du graphe extrait avec des passages des documents bruts. Il est adapté aux questions liées à des entités spécifiques.
- **Le mode de recherche global (*Global Search*) :** Il exploite les rapports de communautés selon une approche *map-reduce*, chaque rapport produisant une réponse partielle ensuite synthétisée. Ce mode vise plutôt les requêtes de synthèse ou d'analyse globale.

Le challenge impose de restituer les références sous forme de paires *(document, page)*. Nous avons donc adapté la sortie de Microsoft GraphRAG en exploitant les métadonnées de page conservées lors de l'indexation. Cette contrainte nous a conduits à n'utiliser que le mode local : le mode global,

1. https://microsoft.github.io/graphrag/

fondé sur les rapports de communautés, peut agréger indirectement un trop grand nombre de passages du corpus. Ainsi, une requête mobilisant deux ou trois résumés de communauté peut impliquer des milliers de passages, ce qui rend difficile l'identification d'un ensemble raisonnable et pertinent de pages de preuve.

Indexation. En entrée du pipeline d'indexation, nous fournissons un ensemble de fichiers JSON, chaque fichier correspondant à un document PDF. Chaque fichier JSON est structuré sous la forme `[{"text": ..., "page": ...}, ...]` afin de regrouper les textes par page, et nous avons configuré Microsoft GraphRAG pour qu'il conserve les métadonnées sur les numéros de page lors de l'indexation. De cette manière, le découpage est effectué à l'intérieur de chaque page, ce qui permet d'associer chaque chunk à un document et à un unique numéro de page. Le système découpe les textes en fenêtres de 600 tokens avec un recouvrement de 100 tokens. Les 19 036 chunks obtenus traversent ensuite un pipeline basé sur `google/gemma-4-31B-it` pour l'extraction d'entités et de relations, et la création des rapports de communautés [2]. Les sorties sont stockées dans des fichiers Parquet et des index vectoriels sont créés dans LanceDB à l'aide de `Qwen/Qwen3-Embedding-8B`.

Requête. Pour une question donnée, nous utilisons le mode de recherche local de Microsoft GraphRAG pour générer une réponse à l'aide de `google/gemma-4-31B-it` servi via vLLM, avec la température par défaut fixée à 0. Nous récupérons ensuite le contexte ayant servi à la génération et appliquons un post-traitement fondé sur les métadonnées conservées lors de l'indexation afin d'identifier les documents sources et les numéros de page associés aux chunks récupérés.

3.3 Hybrid RAG avec Text2Graph

Nos runs 3 et 4 partent du même constat : le Vector RAG fournit des pages pertinentes, mais il peut manquer certains liens relationnels lorsque la question dépend d'entités ou de contraintes dispersées dans plusieurs documents. Nous avons donc construit deux variantes hybrides qui conservent le retriever Vector RAG comme socle et ajoutent une couche de recherche dans un graphe de connaissances extrait automatiquement à partir des chunks structurels.

Index de triplets. Les runs hybrides utilisent l'index de triplets décrit en section 2. À ce stade, chaque fait est déjà verbalisé et rattaché à son chunk source, au document PDF, aux pages couvertes et aux intervalles `page_spans`. À la requête, cet index de triplets est interrogé avec BM25. Le texte indexé concatène la verbalisation du triplet, les libellés du sujet, du prédicat et de l'objet, ainsi que le nom du PDF. Pour chaque question, nous conservons les 80 meilleurs triplets. Ces triplets ne sont pas directement soumis tels quels au format final du challenge, mais ils servent à construire des éléments de preuve supplémentaires, rattachés à des pages de documents.

Fusion avec le Vector RAG. Les deux variantes hybrides reprennent les preuves du run 1 (Vector RAG), mais ne gardent que les 8 premières pages du système de référence. Elles ajoutent ensuite au plus 4 éléments issus du graphe de connaissances. Dans l'ordre transmis au modèle, les 4 premières preuves Vector RAG sont placées en tête, les preuves issues du graphe sont insérées ensuite, puis les preuves Vector RAG restantes sont ajoutées. La génération reçoit donc au maximum 12 éléments de contexte, et utilise le même modèle et les mêmes paramètres de génération de réponse que le run 1 (Vector RAG).

Run 3 : *chunks de graphe*. Dans cette variante, les triplets servent uniquement d'index de recherche.

2. Nous avons volontairement utilisé le même LLM pour l'approche Microsoft GraphRAG et pour l'approche Text2Graph de Lettria

Les 80 triplets les mieux classés sont regroupés par chunk source, puis les chunks sont triés par le meilleur score de leurs triplets associés. Pour chaque chunk retenu, nous récupérons le texte original du chunk, puis nous le redécoupons selon les *page spans* conservés lors du prétraitement afin de produire une preuve rattachée à une page physique. Les pages déjà présentes dans les 8 preuves Vector RAG sont ignorées pour éviter d'ajouter deux fois la même source.

Run 4 : *triplets de graphe*. Dans cette variante, les triplets eux-mêmes sont ajoutés au contexte du modèle. Les triplets classés par BM25 sont regroupés par document, page et chunk, puis chaque élément de preuve contient jusqu'à 5 faits verbalisés associés à ce passage. Le texte transmis au modèle prend la forme d'une courte liste de faits extraits du graphe de connaissances. La provenance reste néanmoins exprimée au niveau *(document, page)* car chaque groupe de faits garde la page du chunk dont les triplets sont issus.

4 Tâche 2 : méthodes testées

4.1 LLM en tant que juge

La tâche 2 consiste à attribuer les phrases d'une réponse existante aux pages candidates fournies dans l'entrée officielle. Nous la formulons comme un jugement contrôlé par LLM (Zheng *et al.*, 2023) : pour chaque question, le modèle reçoit la question, les phrases de la réponse à attribuer et le texte des pages autorisées, puis décide quelles pages soutiennent directement chaque phrase.

Construction du contexte. Chaque paire *(document, page)* de `retrieved[]` est résolue dans le Markdown parsé, puis le contexte est limité aux 12 premières pages candidates et à 30 000 caractères. Dans les runs 1 et 3, seules les pages textuelles sont fournies, sans ajout de triplets de graphe.

Prompt d'attribution. La consigne impose d'utiliser uniquement les pages fournies, d'attribuer une phrase seulement aux pages qui la soutiennent directement, de produire une liste vide lorsqu'aucune page fournie ne la soutient, de ne pas réécrire les phrases et de retourner un JSON contenant, pour chaque `sentence_index`, une liste `attributed_to` de paires `doc_name`, `page`. Une justification courte est conservée dans `reasoning.jsonl` pour inspection manuelle des résultats.

Post-traitement. La réponse JSON du modèle est normalisée avant écriture. Pour chaque phrase officielle, nous recopions l'identifiant `sid` et le texte original de la phrase. Les sources proposées par le modèle sont filtrées par rapport à l'ensemble des pages effectivement présentes dans le contexte : une page non fournie au modèle, mal formatée ou dupliquée est supprimée.

Runs soumis. Le run 1 utilise `google/gemma-4-31B-it` avec une température de 0. Le run 3 conserve le même pipeline et les mêmes paramètres, mais remplace le juge par `Qwen/Qwen3.6-35B-A3B` (Qwen Team, 2026) afin d'évaluer l'effet de la famille de modèles sur l'attribution.

4.2 Graphes de connaissances

Les runs 2 et 4 reprennent le même protocole d'attribution que les runs 1 et 3, mais enrichissent le contexte de chaque page candidate avec des faits issus du graphe de connaissances. Contrairement à

la tâche 1, le graphe ne sert pas à chercher de nouvelles pages : il complète uniquement les pages déjà présentes dans la liste `retrieved[]` fournie par l'entrée officielle.

Sélection des triplets. Nous utilisons le même index JSONL de triplets que pour les runs hybrides de la tâche 1. Seuls les triplets dont la provenance correspond aux couples *(document, page)* officiels sont chargés. Pour chaque page, ils sont dédupliqués, triés par recouvrement lexical avec la question et les phrases à attribuer, puis limités à 12 triplets.

Contexte fourni au juge. Chaque source contient le texte de la page puis, si disponibles, les triplets verbalisés sélectionnés. Les limites de 12 pages et 30 000 caractères, les consignes d'attribution et le format JSON attendu restent identiques à la section 4.1.

Runs soumis. Le run 2 utilise `google/gemma-4-31B-it` avec le contexte textuel enrichi par les triplets du graphe. Le run 4 utilise le même enrichissement, les mêmes limites de contexte et le même post-traitement, mais avec `Qwen/Qwen3.6-35B-A3B`.

4.3 Méthode ensembliste par vote majoritaire

Le cinquième et dernier run de la tâche 2 combine les attributions produites par les quatre runs précédents : Gemma avec texte seul, Gemma avec texte et triplets, Qwen avec texte seul, et Qwen avec texte et triplets. L'objectif est de privilégier les sources stables, c'est-à-dire les paires *(document, page)* retrouvées malgré le changement de famille de modèle ou malgré l'ajout d'informations issues du graphe de connaissances.

Alignement des entrées. Avant tout vote, le script vérifie que les quatre fichiers d'attribution portent sur les mêmes identifiants de questions et sur les mêmes phrases. Pour chaque question, les phrases doivent avoir le même `sid` et le même texte dans les quatre runs.

Vote par source. Le vote est effectué indépendamment pour chaque phrase. Une source correspond à une paire *(document, page)* proposée par au moins un des quatre runs. Pour chaque source candidate, nous comptons le nombre de runs qui l'ont sélectionnée et le nombre de familles de modèles représentées. En l'occurrence, les deux runs Gemma appartiennent à une même famille, et les deux runs Qwen à une autre. La règle principale conserve une source si elle vérifie au moins l'une des deux conditions suivantes : elle est soutenue par les deux familles de modèles, ou elle obtient au moins 3 votes sur 4. Cette règle permet de garder une source confirmée par Gemma et Qwen même lorsqu'elle n'apparaît que dans deux runs, tout en conservant aussi les sources très majoritaires au sein des autres variantes. Si aucune source ne passe cette règle principale, nous conservons uniquement la source arrivée seule en tête lorsqu'elle obtient au moins 2 votes sur 4. Si plusieurs sources sont ex aequo dans ce cas, aucune source n'est ajoutée.

Ordonnancement et sortie. Les sources retenues sont triées par nombre de votes décroissant, puis par nombre de familles de modèles décroissant. En cas d'égalité, nous utilisons la première position observée dans les listes d'attribution, puis l'ordre lexical du document et le numéro de page pour obtenir une sortie entièrement déterministe. Au plus 3 sources sont conservées par phrase. La sortie reprend ensuite le `sid`, le texte de la phrase et la liste filtrée des sources retenues.

5 Résultats et discussion

Les résultats rapportés dans cette section correspondent aux scores communiqués par les organisateurs avant l'adjudication finale.

5.1 Tâche 1

Le tableau 1 présente les scores des quatre runs soumis pour la tâche 1, ainsi que la moyenne des runs de l'ensemble des participants. Le run 1, correspondant au Vector RAG, obtient les meilleurs scores sur toutes les métriques rapportées. Il atteint notamment 0,7039 en LLMAAJ (LLM-As-A-Judge), 0,5658 en MRR@10, 0,5866 en Recall@10 et 0,4971 en NDCG@10, avec un écart positif à la moyenne des participants sur chacune de ces métriques.

Run	LLMAAJ	MRR@10	Recall@10	Top 1	NDCG@10	QBERT	QPARA
Run 1 – Vector RAG	**0,7039**	**0,5658**	**0,5866**	**0,4757**	**0,4971**	**0,7469**	**0,8738**
Run 2 – Microsoft GraphRAG	0,5166	0,2103	0,2865	0,1262	0,2002	0,7046	0,7379
Run 3 – Hybride avec chunks de graphe	0,6922	0,5568	0,5565	**0,4757**	0,4807	0,7454	0,8447
Run 4 – Hybride avec triplets de graphe	0,6859	0,5571	0,5565	**0,4757**	0,4808	0,7376	0,8544
Moyenne participants	0,5910	0,4582	0,4627	0,3789	0,4072	0,6974	0,8311

TABLE 1 – Résultats préliminaires des runs soumis pour la tâche 1.

Les deux variantes hybrides restent au-dessus de la moyenne sur la plupart des métriques, mais n'améliorent pas le Vector RAG. Leur Recall@10 plus faible suggère que les preuves issues du graphe n'ont pas compensé la perte de certaines pages textuelles dans les premiers rangs, faute de réordonnancement final conjoint. Le run Microsoft GraphRAG obtient des scores de recherche nettement plus faibles, ce qui suggère que son mode local, bien qu'adapté à l'interrogation d'entités et de relations, s'aligne moins bien avec l'exigence du challenge consistant à restituer un petit ensemble de pages précisément identifiées.

5.2 Tâche 2

Le tableau 2 présente les scores d'attribution pour les cinq runs soumis. Le meilleur run est le juge Gemma avec contexte textuel seul, qui obtient un F1 d'attribution de 0,681, soit un écart de +0,113 par rapport à la moyenne des participants et de +0,084 par rapport à la médiane.

Run	P1	R1	F1 attr.
Run 1 - Gemma, texte	**0,548**	0,902	**0,681**
Run 2 - Gemma, texte + graphe	0,541	0,912	0,679
Run 5 - Ensemble inter-familles	0,470	0,922	0,623
Run 3 - Qwen, texte	0,430	**0,931**	0,588
Run 4 - Qwen, texte + graphe	0,417	0,882	0,566
Moyenne participants	0,487	0,727	0,568
Médiane participants	0,501	0,789	0,597

TABLE 2 – Résultats préliminaires des runs soumis pour la tâche 2.

Les scores de la tâche 2 montrent un comportement orienté rappel : tous les runs atteignent un R1 élevé, mais les variantes Qwen et le vote ensembliste du run 5 perdent davantage en précision. L'ajout de triplets au contexte Gemma augmente légèrement le rappel, de 0,902 à 0,912, mais réduit la précision de 0,548 à 0,541, ce qui suffit à diminuer légèrement le score F1. Avec Qwen, l'enrichissement par triplets dégrade simultanément la précision, le rappel et le F1. Ces résultats indiquent que les triplets verbalisés peuvent fournir des indices utiles, mais qu'ils introduisent aussi un bruit contextuel susceptible d'encourager des attributions trop larges.

6 Outil de visualisation

Afin de vérifier les runs soumis et de faciliter l'inspection manuelle des résultats, nous avons développé un outil local de visualisation des sorties RAG. L'outil est servi par un serveur HTTP local, lancé avec la commande `uv run –project tools/viewer rag-viewer`. Il lit le corpus parsé en Markdown, les PDF bruts, le répertoire `runs/` et les entrées officielles pour la tâche 2, puis expose une interface web sur le port 8765.

FIGURE 1 – Interface de l'outil de visualisation

Pour la tâche 1, l'interface permet de charger un fichier `task1.json`, de filtrer les questions et d'inspecter la réponse générée avec les pages retrouvées. Pour la tâche 2, elle charge l'entrée officielle et le fichier `task2.json`, puis affiche les phrases, leurs sources attribuées et les pages candidates. Dans les deux cas, un clic sur une source résout le PDF correspondant, reconstruit le texte de la page depuis le Markdown et ouvre le PDF brut. Nous ouvrons le code source de cet outil de visualisation pour la communauté.

7 Estimation du coût carbone

Nous avons estimé le coût énergétique et les émissions associées à l'ensemble des runs soumis avec la méthodologie Green Algorithms (Lannelongue *et al.*, 2021). Cette estimation couvre la préparation des documents, l'indexation et l'inférence des deux tâches. Elle ne correspond donc pas au coût marginal d'une seule requête, mais au coût global de production de la soumission finale. Les modèles utilisés pour les runs soumis sont des modèles à poids ouverts servis localement via vLLM ou Text Embeddings Inference. Aucun entraînement ou fine-tuning n'a été effectué dans le cadre du challenge.

Phase	**Énergie estimée**	**Émissions estimées**
Préparation / indexation des documents	47,94 kWh	2,46 $kgCO_2e$
Entraînement	0,00 kWh	0,00 $kgCO_2e$
Inférence, tâche 1	5,70 kWh	0,29 $kgCO_2e$
Inférence, tâche 2	11,64 kWh	0,60 $kgCO_2e$
Total	65,28 kWh	3,35 $kgCO_2e$

TABLE 3 – Estimation agrégée du coût énergétique et carbone des runs soumis.

La phase la plus coûteuse est la préparation et l'indexation des documents, avec 47,94 kWh, soit environ 73 % de l'énergie totale estimée. Cette phase regroupe notamment le parsing des 375 documents du corpus, la construction des index textuels, la génération des embeddings et la préparation des graphes de connaissances. L'inférence de la tâche 2 est plus coûteuse que celle de la tâche 1 car elle exécute plusieurs variantes d'attribution sur dix sous-ensembles d'entrées avec deux modèles LLM.

8 Conclusions et travaux futurs

Nous avons présenté ELO-GRAG, un système soumis au challenge EvalLLM 2026. Pour la tâche 1, le Vector RAG obtient nos meilleurs scores grâce à la combinaison d'un index dense, d'un index BM25, d'une fusion RRF pondérée et d'une sélection dynamique des pages. Les variantes à base de graphe restent compétitives, mais n'améliorent pas le classement des sources par rapport au run vectoriel, tandis que Microsoft GraphRAG s'adapte moins bien à une évaluation fondée sur quelques pages de preuve précisément identifiées.

Pour la tâche 2, le meilleur compromis est obtenu par un juge Gemma utilisant uniquement le texte des pages candidates. L'enrichissement par triplets augmente légèrement le rappel avec Gemma, mais réduit la précision et n'améliore pas le F1, ce qui confirme que le risque principal est la sur-attribution. Le modèle retient parfois une page liée à la réponse, mais pas nécessairement une page soutenant directement la phrase évaluée.

La principale leçon est que la qualité d'un système RAG pour ce corpus dépend fortement de la sélection et de la calibration des sources, au moins autant que du modèle génératif utilisé. Les travaux futurs porteront donc sur un réordonnancement conjoint des preuves vectorielles et graphe, une calibration plus stricte des juges d'attribution, et une extraction de triplets plus sélective, par exemple guidée par une ontologie.

Remerciements

Ce travail a été partiellement financé par le programme i-Demo de la Banque Publique d'Investissement (Bpifrance) au sein du projet LettRAGraph (projet DOS0256163/00).

Références

CORMACK G. V., CLARKE C. L. A. & BUETTCHER S. (2009). Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 758–759 : Association for Computing Machinery. DOI : 10.1145/1571941.1572114.

EDGE D., TRINH H., CHENG N., BRADLEY J., CHAO A., MODY A., TRUITT S., METROPOLITANSKY D., NESS R. O. & LARSON J. (2025). From Local to Global : A Graph RAG Approach to Query-Focused Summarization. DOI : https ://doi.org/10.48550/arXiv.2404.16130.

GOOGLE DEEPMIND (2026). Gemma 4 Model Card.

KWON W., LI Z., ZHUANG S., SHENG Y., ZHENG L., YU C. H., GONZALEZ J. E., ZHANG H. & STOICA I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

LANNELONGUE L., GREALEY J. & INOUYE M. (2021). Green Algorithms : Quantifying the Carbon Footprint of Computation. *Advanced Science*, **8**(12), 2100707. DOI : 10.1002/advs.202100707.

LEWIS P., PEREZ E., PIKTUS A., PETRONI F., KARPUKHIN V., GOYAL N., KÜTTLER H., LEWIS M., YIH W., ROCKTÄSCHEL T., RIEDEL S. & KIELA D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *34th International Conference on Neural Information Processing Systems (NeurIPS)*.

LISENA P., PLU J., ESCOBAR O., TROUILLEZ E. & TRONCY R. (2026). Pitfalls in AI-Generated Ontologies : Strategies for Detection and Mitigation. In *International Workshop on LLM-driven Knowledge Graph and Ontology Engineering (LLM4KG-OE)*, Dubrovnic, Croatia.

MELI SONGUON C., CHABOT Y. & TRONCY R. (2026). Les systèmes GraphRAG en pratique : vers une taxonomie des questions, des méthodes et défis d'évaluation. In *Atelier sur l'évaluation des modèles génératifs (LLM), le RAG et challenges (EvalLLM) colocalisé avec CORIA-TALN*, Nantes, France.

PENG B., ZHU Y., LIU Y., BO X., SHI H., HONG C., ZHANG Y. & TANG S. (2025). Graph Retrieval-Augmented Generation : A Survey. *ACM Transactions on Information Systems*. DOI : 10.1145/3777378.

PLU J., ESCOBAR O., TROUILLEZ E., GAPIN A., LISENA P., EHRHART T. & TRONCY R. (2025). Text2KGBench-LettrIA : A refined benchmark for Text2Graph systems. In CEUR, Éd., *International Workshop on Knowledge Base Construction from Pre-trained Language Models (KBC-LM)*, Nara, Japan.

QWEN TEAM (2026). Qwen3.6-35B-A3B. Hugging Face model card.

SINGER-VINE J. & PDFPLUMBER CONTRIBUTORS T. (2026). pdfplumber. Software.

SMOCK B., PESALA R. & ABRAHAM R. (2022). PubTables-1M : Towards Comprehensive Table Extraction From Unstructured Documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 4634–4642. DOI : 10.1109/CVPR52688.2022.00459.

TRAAG V. A., WALTMAN L. & VAN ECK N. J. (2019). From Louvain to Leiden : guaranteeing well-connected communities. *Scientific Reports*, **9**(1). DOI : 10.1038/s41598-019-41695-z.

WANG P., XU B., ZHANG L., WANG S., DU M., ZHU C. & MAO Z. (2026). WildGraphBench : Benchmarking GraphRAG with Wild-Source Corpora. DOI : https ://doi.org/10.48550/arXiv.2602.02053.

ZHANG Y., LI M., LONG D., ZHANG X., LIN H., YANG B., XIE P., YANG A., LIU D., LIN J., HUANG F. & ZHOU J. (2025). Qwen3 Embedding : Advancing Text Embedding and Reranking Through Foundation Models. DOI : 10.48550/arXiv.2506.05176.

ZHAO Z., KANG H., WANG B. & HE C. (2024). DocLayout-YOLO : Enhancing Document Layout Analysis through Diverse Synthetic Data and Global-to-Local Adaptive Perception. DOI : 10.48550/arXiv.2410.12628.

ZHENG L., CHIANG W.-L., SHENG Y., ZHUANG S., WU Z., ZHUANG Y., LIN Z., LI Z., LI D., XING E. P., ZHANG H., GONZALEZ J. E. & STOICA I. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36.