Sur la génération de grands graphes de connaissances synthétiques à l'aide de modèles de langage (LLM) via une approche Ontology2Graph

Philippe Dooze[1], Lionel Tailhardat[1], Ovidiu Pascal[1]

[1] Orange Research, France

philippe.dooze@orange.com, lionel.tailhardat@orange.com, ovidiu.pascal@orange.com

Résumé

Les graphes de connaissances synthétiques (GCS) sont utilisés dans divers domaines, comme l'ingénierie et la médecine. Plusieurs approches algorithmiques de génération existent mais leur mise en œuvre est complexe. Ce travail explore l'utilisation de grand modèles de langage (LLMs) selon une approche Ontology2Graph pour vérifier s'ils facilitent la création de GCS. Nos expérimentations soulignent l'importance du contrôle syntaxique et ontologique ainsi que la stratégie de fusion de graphes post-génération pour contourner la limitation de tokens en sortie des LLMs.

Mots-clés

Génération de graphes, Graphes de connaissances, Données fictives, Ontologie, Ontology2Graph, IA générative, LLM, Fusion de graphes.

Abstract

Synthetic knowledge graphs (SKG) are used across various fields, such as engineering and medicine. Although several algorithmic approaches for graph generation exist, their implementation is complex. Our work explores the use of Large Langage Models (LLMs) through an Ontology2Graph approach to determine how they ease the creation of SKG. Our experiments highlight the importance of syntax and ontologic controls and the need for post-generation graph fusion strategies to overcome LLM token limits.

Keywords

Graph generation, Knowledge graphs, Synthetic dataset, Ontology, Ontology2Graph, Generative AI, LLM , Graph fusion.

1 Introduction

Les GCS – des graphes de connaissances structurés par un vocabulaire formel représentatif d'un domaine de discours et composés de données fictives – offrent des avantages pratiques dans divers domaines comme la cybersécurité [21] ou la médecine [15]. Ils permettent de contourner les problèmes de confidentialité et de disponibilité limitée des données. Cela facilite, par exemple, la validation de pipelines d'IA ou la réalisation d'études de faisabilité pour des projets en ingénierie des connaissances.

Plusieurs approches algorithmiques sont envisageables pour générer ces graphes (règles de réécriture [6], modèles stochastiques [11], apprentissage profond [28, 2]). Leur mise en œuvre reste cependant complexe, principalement en raison de la nécessité d'une forte adhérence entre l'approche choisie et le domaine de données, ainsi que de l'utilisation de pipelines souvent sophistiqués. Simplifier la création de GCS pourrait donc démocratiser leur usage et accélérer le développement d'applications basées sur les graphes de connaissances.

Les LLMs récents sont capables d'interpréter une description générale d'un problème et y répondre dans un format spécifique grâce au prompt engineering [4]. Leur utilisation offre la promesse de simplifier le procédé de génération de GCS, mais peu de travaux scientifiques abordent directement ce sujet. Les LLMs sont plutôt employés dans des tâches connexes : ingénierie des connaissances [17, 19], extraction d'ontologies (ontology learning), construction de graphes de connaissances à partir de données textuelles ou semi-structurées (Text2Graph) [3, 14, 25], génération de pipelines de construction de graphes de connaissances [10].

Dans ce travail, nous explorons l'utilisation des LLMs via une approche **Ontology2Graph**, afin d'évaluer leur capacité à produire des GCS. Cette méthode consiste à solliciter un LLM pour générer un GCS à partir d'une ontologie et de consignes fournies en entrée. Nous étudions notamment : **QR. 1** la capacité des LLMs à générer de grands graphes (en ordre et taille) [1], et **QR. 2** la cohérence de ces graphes avec l'ontologie en question et le domaine associé, notamment en ce qui concerne l'instanciation de classes d'objets sémantiquement valides. A travers notre protocole expérimental, nous mettons en évidence l'importance conjointe du choix du LLM et du prompt, ainsi que l'intérêt d'un contrôle syntaxique/ontologique et d'une stratégie de fusion de graphes post-génération.

Le reste de ce document s'articule comme suit : La Section 2 présente les outils et travaux connexes. La Section 3 détaille l'approche *Ontology2Graph*. La Section 4 expose nos expériences et résultats. Enfin, la Section 5 conclut par

1. Nombre de nœuds et nombre d'arêtes (ou d'arcs) du graphe.

une synthèse de nos travaux et aborde les perspectives futures. Le code source d'*Ontology2Graph*, ainsi que les données de test, sont disponibles en open source à l'adresse suivante : https://github.com/Orange-OpenSource/Ontology2Graph.

2 Travaux connexes

Plusieurs méthodes permettent de générer des GCS (Table 1). On distingue les approches produisant directement des graphes de connaissances de celles produisant des graphes sans formalisme spécifique, nécessitant une transformation ultérieure pour obtenir un GCS.

Graphes de connaissances. L'approche stochastique, comme celle de PyGraft [18], permet de créer de grands GCS respectant un vocabulaire de référence, mais sans garantir la présence de structures spécifiques au domaine. D'autres méthodes s'appuient sur des modèles d'IA spécialisés, tels que les auto-encodeurs de graphes variationnels et les modèles de diffusion [15, 2]. Ces modèles, bien qu'efficaces, nécessitent des données d'entraînement et une infrastructure complexe et coûteuse pour leur développement et leur utilisation. Enfin, certaines approches utilisent des LLMs. Par exemple, Shuran Fu *et al.* [8] génèrent un GCS en étendant un graphe de connaissances réel fourni en entrée, mais ne permettent pas de partir d'une simple ontologie. GraphGPT [2], quant à lui, est un outil de type "Text2Graph" où un LLM détecte les relations entre les entités d'un texte et produit les triplets correspondants, sans toutefois reconnaître les entités ni structurer le graphe en rapport à une ontologie de référence.

Graphes de données sans formalisme spécifique. Des bibliothèques comme `networkx` [3] et `igraph` [4] offrent divers générateurs de graphes basés sur des modèles paramétriques stochastiques, tels que Erdős-Rényi pour l'attribution d'arêtes entre des paires de nœuds [5] avec une probabilité fixe, ou Barabási-Albert pour des distributions de degrés sans échelle des nœuds. Ces générateurs permettent de produire rapidement des graphes de données, mais ceux-ci ne peuvent pas être qualifiés nativement de graphes de connaissances. Pour les transformer en GCS, il est nécessaire de développer des principes de transformation de données, s'appuyant sur des tables de correspondance avec une ontologie, comme proposé par le composant `neo-semantics` [6] et l'écosystème YARS-PG [23].

Positionnement de l'approche Ontology2Graph. Plutôt que de développer une couche de médiation-transformation entre le générateur de données et l'étape de production du GCS, nous exploitons directement les capacités d'inférence des LLMs en nous inscrivant dans la lignée des méthodes Text2Graph. Deux variantes de Text2Graph sont généralement distinguées à partir des contraintes imposées au LLM [25] : "OIE-based" (Open Information Extraction), qui s'appuie uniquement sur le texte fourni en entrée pour extraire les entités et leurs relations ; "ontology-driven", qui fait intervenir un texte contraint par une ontologie associée. Notre méthode se concentre sur l'exploitation exclusive de l'ontologie fournie au LLM, le forçant alors à puiser dans son corpus d'entraînement des noms de nœuds et des valeurs de littéraux relatif au domaine de discours de l'ontologie. Le GCS généré est ainsi strictement corrélé aux concepts de l'ontologie et aux consignes fournies au LLM. Par conséquent, nous proposons d'appeler cette variante *Ontology2Graph*.

Solution	Méthode	GSF	GDC
Pygraft [18]	Stochastique	✗	✓
FG-difusion [2]	Modèle d' IA	✗	✓
Nikolentzos et al. [15]	spécifique	✗	✓
NetworkX	Stochastique	✓	✗
igraph	et déterministe	✓	✗
GraphGpt	Text2Graph	✓	✗
Shuran Fu et al. [8]	et LLM existant	✗	✓

TABLE 1 – Outils de génération.
Nous distinguons les solutions selon la méthode de génération utilisée et le type de graphe produit : sans formalisme spécifique (GSF) *vs* de connaissances (GDC).

3 L'approche Ontology2Graph

Notre approche exploite un LLM pour créer un GCS à partir d'une ontologie. Plusieurs appels au LLM produisent divers GCS, ensuite fusionnés pour obtenir un GCS plus grand. Les détails de ces étapes sont présentés ci-dessous, avec une vue d'ensemble du processus illustrée en Figure 1.

Génération de GCS par LLM. Nous faisons l'hypothèse que l'ontologie fournit suffisamment d'informations pour que le LLM, grâce à ses données d'entraînement et à un prompt adapté, produise un GCS correctement instancié (càd. répondant à des critères de syntaxe, de cohérence sémantique et d'utilisabilité). Nous utilisons une stratégie de prompt standard, structuré avec : une **description de tâche** (p.ex. "*à partir d'une ontologie, générez un fichier décrivant un nouveau graphe de connaissances basé sur ce schéma*") ; des **instructions** (p.ex. "*les entités doivent provenir de example.com, déclaré comme préfixe*") ; un **contexte** (p.ex. l'ontologie NORIA-O [24]). Le prompt initial P1, représentatif d'une approche zero-shot [7], sert de référence pour nos expérimentations avec deux LLMs représentatifs de l'état de l'art en 2025 (Table 2). L'optimisation du prompt, incluant une approche few-shot [8] et des instructions supplémentaires dans le prompt système [9], est envisagée après l'évaluation initiale de l'adéquation des modèles à notre approche *Ontology2Graph*.

2. https://github.com/varunshenoy/GraphGPT
3. https://networkx.org/documentation/stable/reference/generators.html
4. https://python.igraph.org/en/stable/generation.html
5. Nous employons le terme "nœud" afin d'indifférencier le rapport à la terminologie de la théorie des graphes (sommet) et des graphes de connaissances (entité).
6. https://github.com/neo4j-labs/neosemantics
7. Le modèle prédit la réponse uniquement à partir d'une description en langage naturel de la tâche.
8. Des exemples sont ajoutés au prompt pour guider le modèle.
9. Instructions spécifiques pour chaque tâche, masquées à l'utilisateur.

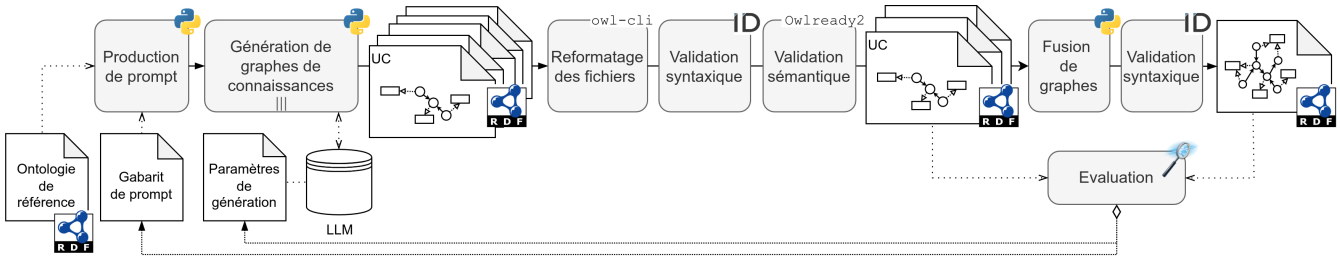


FIGURE 1 – Vue d'ensemble de l'approche Ontology2Graph.
Ce diagramme présente les étapes essentielles de l'approche : un prompt est produit par inclusion d'une ontologie dans un gabarit. Ce prompt est utilisé dans plusieurs appels à un LLM, produisant un GCS différent à chaque fois ; un reformatage est réalisé (`owl-cli`), puis une validation syntaxique (IDLabResearch `TurtleValidator`) et sémantique (`Owlready2`) rejettent les GCS incorrects. Les GCS valides sont fusionnés afin d'augmenter la taille du graphe résultant. L'évaluation (p.ex. test de conformité, panel d'experts) guide l'ajustement manuel des paramètres aux différentes étapes du processus.

Modèle	Capacités	⌈token⌉ entrée	⌈token⌉ sortie
Gpt-4.1-nano Data cutoff : 06/2024 Référence : [16]	Function calling Vision	1 047 576	32 768
Gemini-2.5-flash Data cutoff : 01/2025 Référence : [7]	Function calling Vision Web search Url context Reasoning	1 048 576	65 565

TABLE 2 – LLM utilisés et caractéristiques principales.

Fusion de graphes. En raison des limites de tokens en sortie des LLMs (Table 2), nous anticipons le fait qu'ils produiront des GCS d'ordre et de taille bien moindres qu'espérés pour correspondre à des données réelles (p.ex. le graphe CS-KG 2.0 [5] des publications en sciences informatiques contient 25 millions de nœuds et 67 millions de relations). En effet, un token représentant environ quatre caractères, un LLM comme *Gemini-2.5-flash* peut générer un fichier d'environ 262 260 caractères, soit un maximum de 2622 relations (en supposant 100 caractères par relation).

Des expérimentations préliminaires utilisant P1 et les modèles de la Table 2 confirment cette intuition. Elles suggèrent que la production de grands GCS nécessite la fusion de sous-graphes cohérents entre eux. Ces expériences ont également révélé la présence de nœuds homonymes (càd. identiques de par leur dénomination) dans les séries de graphes générés. Cette homonymie rend inutile l'utilisation de méthodes de mise en cohérence d'entités (p.ex. RiMOM [12] qui combine plusieurs stratégies de mises en cohérence, PARIS [13] à base de méthode probabiliste, ou encore la méthode de plongement de graphes [27]).

Nous utilisons donc une méthode de fusion par égalité stricte de nom : un homonyme apparaissant dans deux graphes $\mathcal{G}_A$ et $\mathcal{G}_B$ distincts sera considéré comme un seul et même nœud dans le graphe résultant $\mathcal{G}$ issu de la concaténation des deux fichiers correspondants ; ce nœud restant lié à la fois avec des nœuds de $\mathcal{G}_A$ et des nœuds de $\mathcal{G}_B$, la concaténation des fichiers réalise l'union des graphes. Enfin, dans le but de générer des graphes $\mathcal{G}$ fusionnés d'ordre et de densité différente, nous appliquons une stratégie de renommage des nœuds avant fusion. Les homonymes sont renommés dans un même fichier, mais différemment d'un fichier à l'autre. Plus le nombre de fichiers avec des nœuds renommés est élevé, plus l'ordre du graphe résultant augmente et sa densité [10] diminue.

4 Expérimentations et résultats

Dans cette section, nous testons l'approche *Ontology2Graph* pour répondre aux questions de recherche **QR. 1** & **QR. 2**. Notre processus de génération et de validation des GCS comprend plusieurs étapes clés. Parmi les instructions du prompt, nous demandons au LLM de produire le GCS en syntaxe Turtle [11]. Chaque fichier généré est reformaté via `owl-cli` [12] pour supprimer les commentaires, simplifier la sérialisation (un couple prédicat-objet par ligne) et uniformiser l'indentation, ce qui facilite l'étape de fusion. Nous vérifions ensuite, pour chaque fichier, la conformité syntaxique avec `TurtleValidator` [13] et nous nous assurons de la cohérence sémantique du GCS à l'aide des raisonneurs HermiT et Pellet [9, 22].
Nous analysons ci-après : l'influence du choix et du paramétrage du LLM (Section 4.1), l'impact du prompt (Section 4.2), l'effectivité de la fusion de graphes (Section 4.3). Enfin, nous procédons à une évaluation qualitative des graphes produits (Section 4.4). Pour ces expérimentations, nous utilisons l'ontologie NORIA-O [24], une ontologie RDFS/OWL conçue pour les réseaux de télécommunication et l'analyse de leur cycle de vie. Ce choix est motivé par notre expertise sur cette ontologie et par l'accès à un panel d'experts afférent pour l'évaluation qualitative.

4.1 LLM et paramètres de génération

Mode opératoire. Nous faisons varier les paramètres T (température) [14] et top_p (nucleus sampling) [15], relatifs à la

10. Pour un graphe dirigé : $\Delta(\mathcal{G}) = \frac{e}{n(n-1)}$, avec n le nombre de nœuds e le nombre d'arrêtes.
11. https://www.w3.org/TR/turtle/
12. https://github.com/atextor/owl-cli
13. https://github.com/IDLabResearch/TurtleValidator
14. T contrôle la diversité et la créativité des réponses d'un modèle d'IA : faible (proche de 0) amène des réponses prévisibles, élevée (proche de 1) amène des réponses plus variées et imprévisibles.
15. Le paramètre "nucleus sampling" (ou "probabilité cumulée de sélection d'un groupe de token") est utilisé dans la prédiction du mot suivant en limitant le nombre de mots probable à un sous ensemble dont la somme

créativité du modèle, afin d'estimer leur impact sur la qualité des graphes produits. Pour un même couple $(T; top\_p)$ et le prompt initial P1, nous demandons la création de dix graphes consécutif à *Gpt-4.1-nano* et à *Gemini-2.5-flash*. Pour chaque ensemble de dix graphes, nous relevons le nombre de graphes ne respectant pas la syntaxe Turtle (F_{err}) et calculons la moyenne du nombre de tokens en sortie. Sur l'ensemble des graphes Turtle sans erreurs, nous calculons de même la moyenne du nombre de nœuds ($\overline{n}$) et de la densité ($\overline{\Delta(\mathcal{G})}$). La Table 3 présente les résultats de ces mesures.

> Given a turtle schema (Noria.ttl) describing classes and relations, provide a Turtle result file that describes a new knowledge graph based on this specific schema. You have to create Nodes linked together via the Relations described in the schema provided. Dismiss any isolated nodes, each nodes must have at least one relation with another node. Entities must come from example.com declared as prefix. Group Entities declaration and Relationships in the same statement and use `rdfs:label` and `rdfs:comment` to provide more clarity on Entities. Never repeat the same information and respect RDF, RDFS, OWL vocabularies and Turtle syntax. Never put explanations at the end of the file. Each prefix used in the nodes declarations must be declared at the beginging of the turtle result file. Be creative.

FIGURE 2 – Prompt initial P1.

T	top_p	F_{err}	$\overline{token_{sortie}}$	$\overline{n}$	$\overline{\Delta(\mathcal{G})}$
		Gpt-4.1-nano			
0,1	0,1	7	1952	20	0,0620
0,1	0,5	7	2128	23	0,0493
0,1	1,0	9	2548	**37**	0,0292
0,5	0,1	8	2038	22	0,0545
0,5	0,5	8	2198	20	0,0515
0,5	1,0	7	1923	25	0,0494
1,0	0,1	9	**5483**	19	0,0760
1,0	0,5	**5**	1930	22	0,0665
1,0	1,0	8	1574	12	0,1023
		Gemini-2.5-flash			
0,1	0,1	**0**	**16 248**	63	0,0230
0,1	0,5	7	9246	50	0,0304
0,1	1,0	7	11 867	**78**	0,0216
0,5	0,1	**0**	15 883	63	0,0230
0,5	0,5	4	12 181	58	0,0254
0,5	1,0	7	11 123	55	0,0296
1,0	0,1	**0**	15 882	63	0,0230
1,0	0,5	7	11 811	66	0,0215
1,0	1,0	5	11 300	68	0,0208

TABLE 3 – Performances du prompt P1.

Résultats pour Gpt-4.1-nano. 75 % (68/90) des fichiers Turtle produits présentent une erreur, le moindre taux d'erreur étant de 50 % pour $(T = 1{,}0; top\_p = 0{,}5)$. Le nombre de tokens en sortie est faible au regard de la limite maximale : $\overline{token_{sortie}} = 2419$ au global, soit 7 % de la limite. Une exception a été observée pour la configuration $(T = 1; top\_p = 0{,}1)$ qui a généré un fichier avec 32 768 tokens (limite du modèle) sur les dix fichiers demandés, ce qui explique la moyenne de tokens plus élevée (5483) pour cette configuration. L'analyse du fichier a cependant révélé un contenu tronqué et a été rejeté par le validateur de syntaxe Turtle. Nous ne remarquons pas de tendance particulière concernant le nombre moyen de nœuds et la densité moyenne des graphes en fonction des variation du couple $(T; top\_p)$.

Résultats pour Gemini-2.5-flash. 41 % (37/90) des fichiers Turtle produits présentent une erreur. Pour $top\_p = 0{,}1$, aucune erreur n'est observée ($F_{err} = 0$), quelque soit la valeur de T. Cependant, il s'avère que cette valeur de top_p conduit le LLM à produire systématiquement seulement deux fichiers différents parmi les dix fichiers générés pour chaque couple $(T = *; top\_p = 0{,}1)$: un avec 16 340 tokens, un autre avec 15 246 tokens. Bien que différents en termes de tokens, ces fichiers ont le même nombre de nœuds et de relations, amenant une densité identique. En incrémentant top_p de 0,1 à 0,5 par pas de 0,1, nous avons constaté que les valeurs adéquates pour une diversité satisfaisante de fichiers (dix fichiers différents) correspondent à $top\_p \geq 0{,}4$. Les valeurs de top_p entre 0,1 et 0,3 sont donc à proscrire.

Par ailleurs, le nombre moyen de nœuds est systématiquement plus élevé pour *Gemini-2.5-flash*, avec une valeur globale $\overline{n} = 63$ (soit un facteur 2,83 entre les deux modèles). Le nombre de tokens est également plus élevé, mais reste loin de la limite du modèle : $\overline{token_{sortie}} = 12\,838$ au global, soit 20 % de la limite. Hormis l'influence des valeurs de $top\_p \in \{0{,}1; \cdots 0{,}3\}$ sur la réponse de *Gemini-2.5-flash*, aucune corrélation n'est observée entre les variations du paramétrage et le nombre moyen de nœuds ou la densité moyenne des graphes.

Discussion. Malgré le caractère probabiliste des modèles, deux tendances ressortent : *Gemini-2.5-flash* génère systématiquement plus de nœuds et moins de fichiers en erreur que *Gpt-4.1-nano* ; d'un point de vue qualitatif, *Gemini-2.5-flash* génère des noms de nœuds plus élaborés, tels que `web_portal_dev` ou `web_portal_prod`, évoquant des dénominations de plateformes informatiques de développement ou de production. Ces types de dénominations n'ont jamais été observés avec *Gpt-4.1-nano*. Pour ces raisons, *Gemini-2.5-flash* sera utilisé pour les expérimentations des sections suivantes.

4.2 Optimisation du prompt

Mode opératoire. Nous itérons à partir du prompt P1 par essai-analyse en cherchant la structure et les consignes qui minimisent les principales erreurs détectées et maximisent l'ordre du graphe (nombre de nœuds). Chaque prompt est testé en respectant les invariants suivants : modèle *Gemini-2.5-flash* ; génération de 20 graphes pour chaque série de tests ; paramètres de créativité $(T = 0{,}1; top\_p = 0{,}4)$. La typologie des prompts est présentée en Table 4. L'ensemble des prompts (utilisateurs de P1 à P4_1, système et ceux utilisés pour créer ou optimiser des prompts utilisateurs) est disponible dans https://github.com/Orange-OpenSource/Ontology2Graph.

Phénomène de troncature et maximisation de l'ordre des GCS. Il arrive parfois que certaines erreurs syntaxiques proviennent de la troncature de la dernière ligne du fichier

des probabilités a atteint le seuil défini par la valeur de *top_p*.

Prompt	Créé manuellement	Optimisé manuellement		Optimisé par IA		Créé par IA
P1	✓	✗		✗		✗
P2	✗	✓	(P1)	✗		✗
P2_1	✗	✓	(P2)	✗		✗
P3	✗	✗		✓	(P2)	✗
P3_1	✗	✗		✓	(P2_1)	✗
P4	✗	✗		✗		✓
P4_1	✗	✓	(P4)	✗		✗

TABLE 4 – Typologie des prompts.

Turtle généré, causée par le dépassement de la limite de tokens en sortie du LLM. Afin de réduire ce phénomène de troncature tout en maximisant l'ordre des graphes nous avons spécifié dans le prompt système l'attente d'une réponse proche des 60 000 tokens de sortie.

Résultats prompt initial P1 : Le prompt P1, déjà utilisé en Section 4, a généré quinze fichiers erronés, dont dix (66,7 %) avec une erreur liée au préfixe `xsd`[16]. La moyenne des "reasoning tokens" est biaisée par un fichier tronqué utilisant 62 915 tokens. Sans cette valeur aberrante, la moyenne est de 10 407 au lieu de 13 033. Nous avons ensuite optimisé P1 manuellement, en séquençant les tâches à effectuer et en explicitant les erreurs à éviter, notamment l'erreur relative au préfixe `xsd`, produisant ainsi le prompt P2.

Résultats prompt P2 : Le prompt P2 a entraîné une augmentation des tokens utilisés (reasoning tokens et text tokens), se traduisant par un nombre moyen de nœuds supérieur à P1 (115). La forte diminution des erreurs relatives au préfixe `xsd` a été contrebalancée par l'émergence d'autres causes d'erreurs et l'augmentation des fichiers tronqués, conduisant à un nombre de fichiers en erreur quasi identique à P1. Ainsi, la plus-value de P2 par rapport à P1 est relativement limitée.

Résultats prompt P2_1 : Le prompt P2 a été modifié manuellement pour réduire les fichiers tronqués, amenant ainsi le prompt P2_1, qui précise le nombre de tokens désirés. Avec huit fichiers en erreur et aucun tronqué, ce nouveau prompt a réalisé une bonne performance, bien que le nombre moyen de nœuds (99) soit plus faible que pour P2.

Résultats pour les prompts P3 et P3_1 : A ce stade de l'étude, nous avons décidé d'arrêter l'optimisation manuelle des prompts au profit d'une optimisation par LLM (*GPT-4.1-mini*) basée sur un *prompt d'optimisation de prompt*. Les prompts P3 et P3_1, correspondant respectivement à l'optimisation de P2 et P2_1, ont généré le nombre moyen de nœuds le plus élevé jusqu'alors (120 et 123). Cependant, avec respectivement 13 et 15 fichiers en erreur, ces résultats ne sont pas satisfaisants.

Résultats pour les prompts P4 : Malgré ces résultats mitigés, nous avons exploré davantage la capacité d'un LLM à améliorer nos prompts. Le prompt P4, généré entièrement par un LLM (*GPT-4.1-mini*) grâce à un *prompt de création de prompts*, est le seul à inclure un exemple de début de graphe attendu. Ses performances surpassent celles des prompts P3, avec le plus grand nombre moyen de nœuds par fichier (128) et un taux de fichiers en erreur relativement faible comparativement aux autres résultats, bien qu'encore élevé (11 fichiers, soit un peu plus de 50 %).

Résultats pour les prompts P4_1 : Nous avons modifié P4 en ajoutant manuellement la consigne d'éviter l'espace de nommage Linked Data Platform[17]. Le préfixe `ldp` représentait la principale cause d'erreur avec P4 (36 % des fichiers en erreur), donnant ainsi naissance au prompt P4_1. Cette modification a entraîné une amélioration significative des résultats, avec une forte diminution du nombre de fichiers en erreur et une augmentation du nombre de nœuds. Avec seulement six fichiers en erreur et un nombre moyen de nœuds de 159, P4_1 affiche les meilleures performances.

Résultats pour le prompt P4_1* : Les résultats précédents, autour de P1 à P4, montrent que la moyenne des tokens de sortie est souvent éloignée de la limite imposée par le modèle, malgré la consigne du prompt système : "*using near 60 000 output tokens to maximise your response*". En supprimant cette consigne (résultats P4_1*), nous observons : une production de tokens de sortie encore plus faible pour P4_1* que pour P4_1 ; aucun effet sur le nombre de fichiers tronqués, qui reste très faible avec ou sans cette consigne. Cette consigne, initialement implémentée pour maximiser les graphes et minimiser les phénomènes de troncature, apporte donc un léger bénéfice sur la production de tokens et incidemment de nœuds, bien que cela reste en deçà de la limitation imposée par le LLM.
Par ailleurs le nombre de fichiers en erreur est toujours plus faible pour P4_1 que pour P4_1*, ce qui est un résultat inattendu, et inexpliqué à ce stade de l'étude étant donné la nature de la consigne. La présence de cette consigne dans le prompt système influe donc positivement sur l'ordre des graphes produits et sur leur qualité syntaxique.
Il est à noter que la moyenne des reasoning tokens de la dernière série de tests P4_1* est biaisée par un résultat utilisant 62 914 tokens. Sans ce résultat aberrant, la moyenne est de 3474, ce qui est cohérent avec les autres résultats de P4_1*

Prompt	$\overline{\text{token reasoning}}$	$\overline{\text{token text}}$	$\overline{\text{token sortie}}$	$\overline{n}$
P1	13 033	11 729	24 781	*94*
P2	17 926	25 013	**42 940**	115
P2_1	14 275	13 487	27 756	99
P3	**23 839**	17 168	38 657	120
P3_1	19 397	21 911	41 325	123
P4	8 416	21 870	30 236	128
P4_1	10 260	**26 624**	36 885	**159**
P4_1*	*2 232*	16 317	18 559	*94*
P4_1	4 619	19 725	24 345	144
P4_1*	3 186	13 044	*16 731*	102
P4_1	6 503	19 076	25 579	134
P4_1*	6 446	*11 564*	18 010	106

TABLE 5 – Tokens et nombre de nœuds par prompt.
Le nombre moyen de nœuds ($\overline{n}$) est calculé sur les fichiers syntaxiquement corrects.

Synthèse des résultats. Les résultats obtenus avec les différents prompts sont résumés dans la Table 5 pour la production de nœuds et de tokens, et la Table 6 pour la confor-

16. http://www.w3.org/2001/XMLSchema#

17. Préfixe `ldp` : http://www.w3.org/ns/ldp#

Tu es un expert en modélisation de graphes de connaissances RDF/OWL et en syntaxe Turtle. Ta mission est de générer un graphe de connaissances complet, dense, conforme et connecté, en utilisant exclusivement le vocabulaire NORIA défini ci-dessous. Consignes précises :

1. Respect strict du vocabulaire NORIA : Utilise uniquement les classes, propriétés, préfixes et concepts définis dans le vocabulaire NORIA fourni. Ne crée aucune entité, propriété ou préfixe hors de ce vocabulaire.
2. Conformité syntaxique parfaite : Le graphe doit être valide en syntaxe Turtle, sans erreurs ni entités tronquées. Chaque triplet doit être complet et correctement formé.
3. Complétude des entités : Pour chaque instance créée, fournis un ensemble complet de descripteurs pertinents selon le vocabulaire NORIA, notamment : `rdf:type` (classe exacte), `rdfs:label` en anglais (et en français si possible), `rdfs:comment` ou `skos:example` pour enrichir la description, relations avec d'autres entités via les propriétés du vocabulaire (ex : `noria:resourceManagedBy`, `noria:applicationModuleOf`, etc.)
4. Interdiction du préfixe `ldp` : le préfixe `ldp` (Linked Data Platform) ne doit pas être utilisé, ni déclaré, ni référencé dans le graphe.
5. Maximisation du nombre d'entités : Génère autant d'entités (nœuds) que possible dans la limite des tokens autorisés, en créant un graphe riche, varié et détaillé.
6. Garanties de connexité et densité : Chaque entité doit être liée à au moins une autre entité via une propriété du vocabulaire NORIA. Le graphe doit présenter une densité raisonnable, avec de nombreuses relations entre entités, reflétant des cas réels d'infrastructures IT et de gestion d'incidents. Évite les entités isolées ou orphelines.
7. Organisation claire : Commence par déclarer tous les préfixes NORIA et standards (`rdf`, `rdfs`, `owl`, `xsd`, `foaf`, etc.) tels que définis dans le vocabulaire. Organise la sortie en blocs logiques par type d'entité (ex : Resources, Applications, TroubleTickets, EventRecords, etc.)

Vocabulaire NORIA (extraits clés) : Classes principales : `noria:Resource`, `noria:Application`, `noria:ApplicationModule`, `noria:Service`, `noria:TroubleTicket`, `noria:EventRecord`, `noria:ChangeRequest`, `noria:Agent` (`foaf:Agent`), `noria:NetworkInterface`, `noria:NetworkLink`, etc. Propriétés importantes : `noria:resourceManagedBy` (Resource → `foaf:Agent`), `noria:applicationModuleOf` (ApplicationModule → Application), `noria:troubleTicketRelatedService` (TroubleTicket → Application/Service), `noria:troubleTicketRelatedParty` (TroubleTicket → `foaf:Agent`), `noria:logOriginatingManagedObject` (EventRecord → Resource/Application), `noria:networkInterfaceOf` (NetworkInterface → Resource), `noria:networkInterfaceConnects` (NetworkInterface → NetworkLink), etc.

Utilise les annotations `rdfs:label`, `rdfs:comment`, `skos:example` pour enrichir les entités.

FIGURE 3 – Prompt P4_1.
Ce prompt est issu du prompt P4 que nous avons amélioré manuellement en supprimant toute les références au préfixe `ldp` afin de minimiser les erreurs de syntaxe constatées avec ce préfixe. L'exemple de graphe en fin de prompt a été supprimé pour des raisons de place ; pour plus de détails, consulter le dépôt open source *Ontology2Graph*. Générés via *GPT-4.1-mini* grâce à un *prompt de création de prompts* écrit en Français, les prompts P4 et P4_1 sont en Français contrairement aux autres prompts de l'étude rédigés en Anglais. La section "Vocabulaire NORIA (extrait clés)" des prompts P4 ne remplace pas l'ontologie, toujours fournie intégralement au LLM, quel que soit le prompt utilisé.

mité syntaxique, incluant un décompte et la répartition des principales erreurs syntaxiques.

Concernant les erreurs de syntaxe liées à un préfixe, deux cas distincts sont à considérer. Le premier concerne l'espace de nommage XMLSchema (préfixe `xsd`), utilisé pour la déclaration d'un littéral. Lorsque la valeur du littéral n'est pas encadrée par des guillemets, une erreur du type `Unexpected "10^^xsd:int" on line 400` est générée. Le deuxième cas concerne les autres préfixes, pour lesquels la déclaration d'un nœud n'est pas correctement libellée. Les autres erreurs fréquemment rencontrées incluent l'utilisation de préfixes non définis dans le préambule du graphe (p.ex. `ldp`), ainsi que l'oubli du préfixe dans la déclaration d'un nœud. Le validateur de syntaxe arrêtant la vérification à la première erreur détectée, la Table 6 répertorie le nombre de fichiers pour lesquels au moins une erreur a été détectée.

Discussion. Il apparaît clairement que la qualité du prompt utilisateur est essentielle pour maximiser le nombre de fichiers syntaxiquement corrects et avec un nombre élevé de nœuds. La meilleure stratégie consiste à demander à un LLM de générer un prompt visant à générer un GCS, puis à l'optimiser manuellement si besoin. Cette approche a été utilisée avec le prompt P4_1, qui a généré quatorze graphes syntaxiquement corrects sur les vingt demandés (70 %) avec une moyenne de 159 nœuds par graphe lors de sa première utilisation. Ces résultats positifs ont été confirmés par deux autres séries de tests. De plus, un prompt système incitant le LLM à générer un nombre de tokens de sortie proche de sa limite autorisée permet d'augmenter l'ordre des graphes produits, sans toutefois s'approcher de cette limite, et réduit le nombre de fichiers avec des erreurs syntaxiques. Ce dernier point, non expliqué à ce stade de l'étude, nécessite une vérification de sa reproductibilité avec d'autres LLMs, notamment ceux utilisés dans les solutions décrites en Section 2.

4.3 Fusion des graphes

Algorithme. La première étape consiste, à partir d'un ensemble de graphes $\mathcal{G} = \{\mathcal{G}_1, \cdots \mathcal{G}_N\}$ produits par un LLM, à construire un ensemble $\mathcal{H} = \{(v, occ_v)\}$, où $occ_v, \in [1, \cdots occ_{top}]$, désigne le nombre de graphes dans lesquels le nœud v apparaît au moins une fois. On note $occ_{top} = \max_v occ_v$ la valeur maximale observée. Dans le cas extrême où un même nœud apparaît dans les N graphes de $\mathcal{G}$, nous avons donc $occ_{top} = N$.

Nous procédons ensuite par itérations successives en renommant, dans un seul fichier, les nœuds v dont la valeur occ_v est égale à la valeur maximale de l'ensemble $\mathcal{H}$ (à la première itération occ_{max} = occ_{top}). Chaque itération a pour effet de décrémenter de 1 la valeur de occ_{max} dans $\mathcal{H}$ et de générer un nouvel ensemble $\mathcal{G}occ_{max}$ de N graphes qui, une fois fusionnés, amène à un graphe résultant ayant un ordre et une densité propre.

Résultats. La Table 8 regroupe les densités et nombre de nœuds résultants de la concaténation des 14 fichiers, décrits en Table 7, suivant la décroissance de occ_{max} de 14 à 1. Pour occ_{max} = occ_{top}= 14, en l'absence de renom-

	Fichier tronqué	Erreur de syntaxe	Erreur de syntaxe liée à un préfixe	Préfixe non défini ou absent	F_{err}
P1	1	**0**	xsd(10), kos(3)	ldp(1)	15
P2	5	2	foaf(1), kos(2), xsd(2), noria(1)	absent(1)	14
P2_1	**0**	1	xsd(2), org(1), noria(2)	devopsprod(1), absent(1)	8
P3	**0**	1	xsd(4), kos(1), noria-kos(7)	-	13
P3_1	2	2	xsd(6), noria-kos(4)	absent(1)	15
P4	1	2	foaf(2)	ldp(4), absent(2)	11
P4_1	2	2	kos-app(1)	absent(1)	6
P4_1*	2	1	foaf(1), voaf(1), vann(1)	absent(5)	11
P4_1	1	1	xsd(1)	absent(1)	**4**
P4_1*	1	3	voaf(1)	absent(6)	11
P4_1	**0**	2	kos(1), foaf(2), voaf(1)	-	6
P4_1*	1	3	noria(3), xsd(2)	absent(3)	12

TABLE 6 – Types d'erreurs et total des fichiers en erreur (F_{err}) par prompt.

Graphe #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**Total**
n	214	154	152	161	206	205	167	153	139	161	109	105	161	139	2226
$\Delta(\mathcal{G})$	0,0108	0,0116	0,0140	0,0129	0,0114	0,0103	0,0111	0,0119	0,0153	0,0115	0,0195	0,0167	0,0115	0,0143	

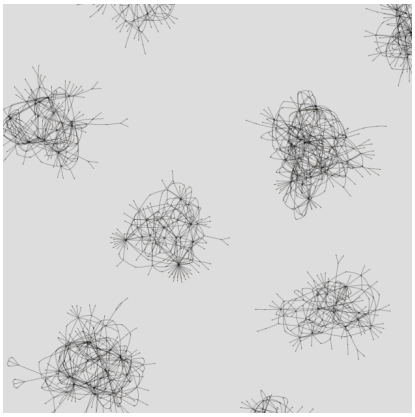
TABLE 7 – Caractéristiques des graphes générés par le prompt P4_1.
Cette table présente les caractéristiques, en nombre de nœuds (n) et densité ($\Delta(\mathcal{G})$), des quatorze graphes syntaxiquement corrects générés par le prompt P4_1 (Section 4.2).

mage d'homonymes, la densité est maximale et le nombre de nœuds est le plus faible (Figure 4b). A l'inverse, lorsque tous les homonymes sont renommés, la Figure 4a illustre clairement une partie des quatorze graphes initiaux générés par le LLM, sans aucune liaison entre eux. Entres ces deux cas de figures, nous avons généré 12 autres graphes, chacun ayant un ordre et une densité distincts.

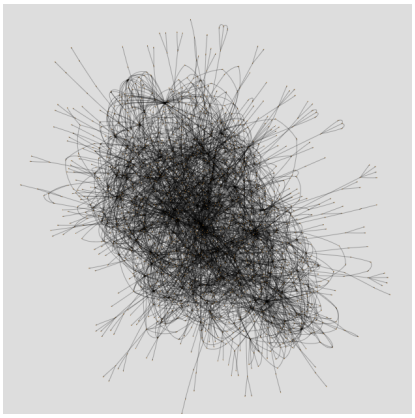
4.4 Evaluation qualitative des graphes

Les GCS générés lors des expérimentations, qu'ils proviennent directement d'un LLM (Sections 4.1 et 4.2) ou du post-traitement de fusion de graphes (Section 4.3), ont tous été validés syntaxiquement. Cependant, cette conformité syntaxique ne garantit pas l'absence d'incohérences sémantiques par rapport à l'ontologie de référence ou à des représentations de situations réelles. Aussi, nous complétons l'évaluation de l'approche *Ontology2Graph* par une analyse qualitative, incluant l'utilisation de *raisonneurs automatiques* et l'évaluation par un *panel d'experts*.

Raisonneurs automatiques. Nous faisons appel aux raisonneurs HermiT et Pellet, au travers de la librairie `owlready2` [18], afin de vérifier la cohérence entre l'ontologie NORIA-O et les graphes générés. Les raisonneurs sont paramétrés avec l'exception `OwlReady Inconsistent Ontology Error` (détection des problèmes de cohérence dans l'ontologie) et la méthode `Default world inconsistent classes` (détection des classes incohérentes). Pour HermiT, qui n'accepte que les ontologies conformes au profil OWL2 DL, nous ajoutons un paramétrage spécifique permettant d'omettre les relations avec un *datatype* incompatible avec l'analyse (p.ex. `xsd:duration`). [19]
Sur l'ensemble des GCS, HermiT et Pellet n'ont dé-



(a) Tous les homonyme renommés



(b) Aucun homonyme renommé

FIGURE 4 – Fusion de graphes et nœuds homonymes.
Ces exemples illustrent l'influence du renommage des homonymes sur la fusion de graphes pour un même jeu de graphes générés (Figures 4a & 4b).

18. https://pypi.org/project/owlready2/
19. https://www.w3.org/TR/owl2-syntax/#Datatype_Maps

occ_{max}	14	13	12	11	10	9	8	7	6	5	4	3	2	1
n	1427	1451	1480	1507	1548	1591	1635	1677	1722	1783	1847	1924	2019	2226
$\Delta(\mathcal{G})$	0,0021	0,0020	0,0020	0,0019	0,0018	0,0017	0,0016	0,015	0,0014	0,0013	0,0013	0,0012	0,0010	0,0009

TABLE 8 – Influence du renommage des nœuds sur le graphe résultant.
Cette table montre l'influence du paramètre occ_{max} sur le nombre de nœuds (n) et la densité ($\Delta(\mathcal{G})$) du graphe résultant de la concaténation des 14 fichiers décrits en Table 7.

tecté aucune classe incohérente. Cependant, ils ont tous deux émis la même alerte : `OwlReady Inconsistent Ontology Error`. Les explications fournies par Pellet ont permis de distinguer trois types d'erreurs derrière cette alerte.

Le premier type d'erreur relève d'une *incohérence logique dans l'ontologie de référence* : la propriété `noria:documentStatusHistory` a pour domaine les classes `noria:TroubleTicket` et `noria:ChangeRequest`, pourtant déclarées disjointes, ce qui implique une contradiction. Le LLM semble ne pas saisir la sémantique de la contrainte de disjonction des classes, générant fréquemment des instances de `TroubleTicket` et `ChangeRequest` avec une même relation `noria:documentStatusHistory` au travers des fichiers générés, ce qui est interdit par l'ontologie NORIA-O. A ce stade, nous attribuons principalement cette erreur à l'ontologie elle-même.

Le second type d'erreur, relatif à la *gestion incorrecte de l'instanciation de relations DatatypeProperty*, a été observé sept fois (Table 9). Pour exemple, la propriété `noria:troubleTicketSupervisionTool`, implémentée avec un `rdfs:range xsd:string`, a été instanciée avec le littéral `"Nagios"@en`. Or, pour rester cohérent avec les spécifications RDF, le LLM n'aurait pas dû annoter le littéral avec un tag de langage dans le graphe, car les tags de langages sont de type "Plain literal" en RDF 1.0 et possèdent le datatype spécifique `rdf:langstring` en RDF 1.1.

DatatypeProperty	F_{err}	F_{lit}	%
`agentInstruction`	14	14	100,0
`logText`	7	7	100,0
`agentWorkingHours`	12	14	85,7
`networkInterfaceDescription`	5	8	62,5
`documentHRef`	6	12	50,0
`networkLinkType`	1	12	8,3
`troubleTicketSupervisionTool`	1	13	7,7

TABLE 9 – Non-conformités sur les DatatypeProperty.
Cette table liste, pour chaque DatatypeProperty incriminée, le nombre de fichiers où le littéral est en erreur (F_{err}), le nombre de fichiers dans lesquels le littéral apparaît (F_{lit}), et le pourcentage F_{err}/F_{lit} correspondant. L'espace de nommage utilisé est `noria`.

Enfin, le troisième type d'erreur, lié à la *génération de littéraux non conformes au type attendu*, n'est apparu qu'une seule fois dans les fichiers générés. Il s'agît de l'utilisation d'une valeur décimale pour une relation `noria:networkInterfaceLaserRxHighPowerWarningThreshold`, alors que l'ontologie NORIA-O spécifie un type `xsd:int`.

Panel d'experts. Pour compléter l'évaluation par une analyse critique des graphes générés par LLM, un panel de six experts en exploitation des réseaux du Groupe Orange [20] a été sollicité. Les participants, aux profils variés (administrateurs réseaux, analystes en cybersécurité, techniciens de supervision, ingénieurs), ont répondu à un sondage évaluant la pertinence (sur quatre niveaux) des relations et des littéraux d'un graphe de test. Le sondage porte sur six types de nœuds clés dans l'exploitation des réseaux, et permet aux répondants d'ajouter des commentaires libres. Le graphe de test utilisé était le graphe #13 de la Table 7, présenté via une interface Web interactive permettant de naviguer et de filtrer le graphe.

Les résultats du sondage, présentés dans la Table 10, indiquent un avis globalement très favorable sur le GCS, avec 42 % d'avis de niveau maximal ($\star\star\star\star$) pour les relations entité-entité générées et 64 % de niveau maximal pour les relations avec un littéral. Les entités décrivant la facette structurelle des réseaux (`noria:NetworkResource`, `noria:NetworkInterface` et `noria:NetworkLink`) reçoivent des avis de niveaux moindres ($\star\star\star$ et $\star\star$). L'analyse des commentaires révèle des attentes spécifiques des experts sur cette facette, comme la notion de réseau multicouche et la distinction entre port physique et logique, qui sont représentables par NORIA-O mais dont l'inférence n'est pas du rôle de l'ontologie elle-même. D'autres commentaires, plus spécifiques aux littéraux, suggèrent un manque d'ancrage du LLM sur des situations réelles, avec l'absence de "*littéraux relatifs au release management des applications et à la résolution d'incidents*".

5 Conclusions et travaux futurs

A travers une approche *Ontology2Graph*, nous avons globalement montré la capacité des LLMs à générer des GCS représentatifs d'un domaine d'application en s'appuyant uniquement sur une ontologie de référence au format RDF-S/OWL passée en argument. Étant donné la facilité de mise en œuvre des LLMs, favorisée par les progrès récents en IA générative et leur adoption massive, nous concluons que ces modèles représentent une solution pertinente pour simplifier la production de GCS, comparativement aux approches existantes. Cette pertinence reste cependant conditionnée par une efficacité globale (coût, ressources) avantageuse. [21]

Dans le détail, nos expérimentations ont révélé que le choix du modèle et la qualité du prompt sont cruciaux pour obtenir des fichiers syntaxiquement corrects et maximiser l'ordre du graphe (**QR. 1**). Un modèle doté de capacités de "raisonnement", comme *Gemini-2.5-flash*, améliore ces aspects. Un prompt utilisateur incluant un extrait du graphe attendu, couplé à un prompt système pour générer

20. https://www.orange.com/

21. Le coût computationnel des LLMs peut être estimé avec des outils comme https://smith.langchain.com/.

Entités	Pertinence des relations entre nœuds					Pertinence des littéraux				
	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆	n.s.p.	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆	n.s.p.
`noria:TroubleTicket`	**50%**	33%	0%	0%	17%	**66%**	17%	0%	0%	17%
`noria:ChangeRequest`	**66%**	17%	0%	0%	17%	**66%**	17%	0%	0%	17%
`noria:Application`	**50%**	17%	17%	0%	17%	**66%**	17%	0%	0%	17%
`noria:NetworkResource`	33%	**50%**	0%	0%	17%	**66%**	0%	17%	0%	17%
`noria:NetworkInterface`	17%	**33%**	**33%**	0%	17%	**50%**	0%	33%	0%	17%
`noria:NetworkLink`	**33%**	17%	**33%**	0%	17%	**66%**	0%	0%	0%	33%

TABLE 10 – Analyse qualitative par un panel d'experts.
Ce tableau présente les résultats, en proportion de réponses par niveau de pertinence du graphe pour chaque catégorie d'objet (entité), du sondage d'un panel de six experts en exploitation des réseaux. Les niveaux de pertinence sont, par ordre décroissant : ⋆⋆⋆⋆ = toutes les relations/littéraux sont pertinent(e)s ; ⋆⋆⋆= les relations/littéraux sont pertinent(e)s mais il manque des informations importantes ; ⋆⋆ = certain(e)s relations/littéraux sont pertinent(e)s mais certain(e)s sont inutiles ou incohérentes ; ⋆ = toutes les relations/littéraux sont inutiles ou incohérent(e)s ; *n.s.p.* = ne se prononce pas.

un nombre de tokens proche de la limite du modèle, renforce la qualité syntaxique et l'organisation du graphe.

Afin de produire de grands graphes en dépassant la limite du nombre de tokens, et donc de dépasser la limite d'environ 140 nœuds pour un GCS, l'utilisation d'un algorithme de fusion de graphes en post-génération est nécessaire. Notre implémentation d'une approche basée sur l'égalité stricte de nom et le renommage séquentiel des nœuds homonymes montre qu'il est aisé d'obtenir un GCS d'environ 2000 nœuds à partir de 14 GCS. Augmenter la taille des graphes en fusionnant davantage de GCS ne semble pas poser de difficultés particulières. Cependant, il serait intéressant d'explorer d'abord comment contrôler la densité des GCS dès leur génération par le LLM (p.ex. via du prompt engineering), ou en post-génération en appliquant une méthode de renommage différente.

Concernant l'adéquation du GCS à un domaine d'application (**QR. 2**), l'analyse qualitative par des raisonneurs automatiques révèle que le processus de génération instancie des classes d'objets de manière cohérente, à condition que leur spécification soit elle-même logiquement cohérente dans l'ontologie. L'instanciation de relations avec des littéraux a engendré plus de non-conformités, principalement liées à des détails de la norme RDF. Pour remédier à ces erreurs, nous envisageons d'améliorer la prise en compte des spécificités des normes par des stratégies de chaînage d'appels au LLM avec des étapes de validation-affinage intermédiaires [26], ou la verbalisation structurée de ces spécificités dans le prompt [20].

Enfin, l'analyse qualitative par un panel d'experts confirme la pertinence globale de la méthode, bien que limitée à l'exploitation des réseaux dans ce travail. Explorer comment introduire des raffinements non dérivables directement de l'ontologie dans les GCS est toutefois nécessaire (par exemple pour NORIA-O [24], des réseaux multicouches ou des anomalies de fonctionnement spécifiques). Plusieurs options sont envisagées pour cela : l'utilisation de LLMs spécialisés [1], l'ajout de consignes dans le prompt – par les utilisateurs ou par extrapolation depuis les questions de compétences [20] accompagnant les ontologies de référence – et l'enrichissement des GCS *a posteriori* avec des règles métier.

Remerciements

Nous tenons à remercier Bertrand Decocq (Orange) pour avoir permis d'organiser le groupe de travail "Graph Generation" en rapport au projet Orange France "Jumeau Numérique des Réseaux". Nous remercions également les membres du panel d'experts : Alassane Samba, François De Turenne, Jérôme Cousty, Mehdi Lebon, Romain Vinel, ainsi que les autres participants anonymes au sondage.

Références

[1] Camille BARBOULE et al. *TelcoLM : Collecting Data, Adapting, and Benchmarking Language Models for the Telecommunication Domain*. 2024. DOI : 10.48550/arXiv.2412.15891.

[2] Adrien BUFORT et Lionel TAILHARDAT. *FGdiffusion : Large-Scale Knowledge Graph Generation Using a Diffusion Approach*. 2025. URL : https://hal.science/hal-05410352.

[3] Harry J. CAUFIELD et al. « Structured Prompt Interrogation and Recursive Extraction of Semantics (SPIRES) : A Method for Populating Knowledge Bases Using Zero-Shot Learning ». In : *Bioinformatics* 40.3 (2024), btae104. DOI : 10.1093/bioinformatics/btae104.

[4] Banghao CHEN et al. *Unleashing the potential of prompt engineering for large language models*. 2025. DOI : 10.48550/arXiv.2310.14735.

[5] Danilo DESSÍ et al. « CS-KG 2.0 : A Large-scale Knowledge Graph of Computer Science ». In : *Scientific Data* 12.1 (2025), p. 964. DOI : 10.1038/s41597-025-05200-8.

[6] Maribel FERNÁNDEZ, Hélène KIRCHNER et Bruno PINAUD. « Strategic Port Graph Rewriting : An Interactive Modelling Framework ». In : *Mathematical Structures in Computer Science* 29.5 (2019), p. 615-662. DOI : 10.1017/S0960129518000270.

[7] Alisa FORTIN et al. *Introducing Gemini 2.5 Flash Image, Our State-of-the-Art Image Model- Google Developers Blog*. 2025. URL : https://developers.googleblog.com/introducing-gemini-2-5-flash-image/.

[8] Shuran FU et al. « Utilizing Language Models For Synthetic Knowledge Graph Generation ». In : *ICLR 2025 Workshop on Navigating and Addressing Data Problems for Foundation Models*. 2025. URL : https://openreview.net/forum?id=IutH9tRtMI.

[9] Birte GLIMM et al. « HermiT : An OWL 2 Reasoner ». In : *Journal of Automated Reasoning* 53.3 (2014), p. 245-269. DOI : 10.1007/s10817-014-9305-1.

[10] Marvin HOFER, Johannes FREY et Erhard RAHM. « Towards Self-Configuring Knowledge Graph Construction Pipelines Using LLMs – A Case Study with RML ». In : *Proceedings of the 5th International Workshop on Knowledge Graph Construction (KGCW)*. T. 3718. Hersonissos, Greece : CEUR-WS.org, 2024. URL : https://ceur-ws.org/Vol-3718/paper6.pdf.

[11] Nicolas HUBERT et al. « PyGraft : Configurable Generation of Synthetic Schemas and Knowledge Graphs at Your Fingertips ». In : *The Semantic Web – 21st International Conference, ESWC 2024, Hersonissos, Crete, Greece, May 26 – 30, 2024, Proceedings, Part II*. 2024. DOI : 10.1007/978-3-031-60635-9_1.

[12] Juanzi LI et al. « RiMOM : A Dynamic Multistrategy Ontology Alignment Framework ». In : *IEEE Transactions on Knowledge and Data Engineering* 21.8 (2009), p. 1218-1232. DOI : 10.1109/TKDE.2008.202.

[13] Fabian M. SUCHANEK, Serge ABITEBOUL et Pierre SENELLART. « PARIS : Probabilistic Alignment of Relations, Instances, and Schema ». In : *Proceedings of the VLDB Endowment* 5, No. 3 (2011). DOI : 10.14778/2078331.2078332.

[14] Rohit Singh NEGI. « Prompt-Based Approach for Knowledge Graph Construction Integrating Quality Assurance Aspects ». Mém. de mast. Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau (RPTU), 2025.

[15] Giannis NIKOLENTZOS et al. « Synthetic Electronic Health Records Generated with Variational Graph Autoencoders ». In : *npj Digital Medicine* 6.1 (2023), p. 83. DOI : 10.1038/s41746-023-00822-x.

[16] OPENAI. *Introducing GPT-4.1 in the API*. 2025. URL : https://openai.com/index/gpt-4-1/.

[17] Bradley P. ALLEN, Lise STORK et Paul GROTH. « Knowledge Engineering Using Large Language Models ». In : *Transactions on Graph Data and Knowledge (TGDK)* 1.1 (2023), 3 :1-3 :19. DOI : 10.4230/TGDK.1.1.3.

[18] Ovidiu PASCAL, Lionel TAILHARDAT et Nicolas HUBERT. *PyGraft-gen*. https://github.com/Orange-OpenSource/pygraft-gen. 2026.

[19] Youssra REBBOUD et al. « Benchmarking LLM-based Ontology Conceptualization : A Proposal ». In : *ISWC 2024, 23rd International Semantic Web Conference*. Baltimore, United States, 2024. URL : https://hal.science/hal-04757816.

[20] Youssra REBBOUD et al. « Can LLMs Generate Competency Questions ? » In : *ESWC 2024, Extended Semantic Web Conference*. Hersonissos, Greece, 2024. URL : https://hal.science/hal-04564055.

[21] James S., Felecia ML. et Donita R. *SAKURA : Synthetic Cyber Knowledge Graph*. https://ncsu-las.org/2024/11/sakura-synthetic-cyber-knowledge-graph/. 2024.

[22] Evren SIRIN et al. « Pellet : A Practical OWL-DL Reasoner ». In : *Journal of Web Semantics* 5.2 (2007), p. 51-53. DOI : 10.1016/j.websem.2007.03.004.

[23] Łukasz SZEREMETA, Dominik TOMASZUK et Renzo ANGLES. « YARS-PG : Property Graphs Representation for Publication and Exchange ». In : *IEEE access : practical innovations, open solutions* 12 (2024), p. 73386-73399. DOI : 10.1109/ACCESS.2024.3403924.

[24] Lionel TAILHARDAT, Yoan CHABOT et Raphaël TRONCY. « NORIA-O : an Ontology for Anomaly Detection and Incident Management in ICT Systems ». In : *Semantic Web – 21st International Conference, ESWC 2024, Hersonissos, Crete, Greece, May 26 – 30, 2024, Proceedings*. 2024. DOI : 10.1007/978-3-031-60635-9_2.

[25] Dali WANG et Mizuho IWAIHARA. « OSKGC : A Benchmark for Ontology Schema-based Knowledge Graph Construction from Text ». In : *Joint Proceedings of the 3rd Workshop on Knowledge Base Construction from Pre-Trained Language Models and the 4th Challenge on Language Models for Knowledge Base Construction (KBC-LM+LM-KBC)*. T. 4041. Nara, Japan : CEUR-WS.org, 2025. URL : https://ceur-ws.org/Vol-4041/paper1.pdf.

[26] Xi YE et al. « SatLM : Satisfiability-Aided Language Models Using Declarative Prompting ». In : *Advances in Neural Information Processing Systems*. T. 36. NeurIPS Proceedings. Curran Associates, Inc., 2023, p. 45548-45580. ISBN : 978-1-7138-9992-1.

[27] Beibei ZHU et al. « A Survey : Knowledge Graph Entity Alignment Research Based on Graph Embedding ». In : *Artificial Intelligence Review* 57.9 (2024), p. 229. DOI : 10.1007/s10462-024-10866-4.

[28] Yanqiao ZHU et al. *A Survey on Deep Graph Generation : Methods and Applications*. 2022. DOI : 10.48550/arXiv.2203.06714.