AlasQA : Système Neurosymbolique de Questions-Réponses sur Graphes de Connaissances

Baptiste Amice*, Peggy Cellier*, Sébastien Ferré*

*Univ Rennes, CNRS, Inria, Insa Rennes, IRISA
prenom.nom@irisa.fr

Résumé. Nous nous intéressons à la tâche d'interrogation d'un graphe de connaissances en langue naturelle. Les graphes de connaissances peuvent être interrogés de façon fiable via des langages formels comme SPARQL. Toutefois cela exige une traduction complexe du langage naturel vers le langage formel. Les grands modèles de langue (LLM) sont capables de répondre directement aux questions en langue naturelle mais n'offrent aucune garantie concernant la validité des réponses qu'ils génèrent. Nous proposons un système neurosymbolique, appelé AlasQA, qui répond à des questions posées en langue naturelle en combinant la fiabilité d'un langage formel comme SPARQL et la puissance des LLMs. La proposition s'appuie sur un outil intermédiaire de construction interactive de requêtes SPARQL. Des expérimentations menées sur les jeux de données QALD et TEXT2SPARQL valident l'intérêt de cette approche hybride.

1 Introduction

Le développement de systèmes de questions-réponses pour l'interrogation de graphes de connaissances (KGQA) est une tâche importante à la croisée du traitement automatique du langage naturel (TALN) et du web sémantique. Contrairement aux moteurs de recherche traditionnels, qui renvoient une liste de documents pertinents, les systèmes de KGQA visent à fournir directement une réponse concise et exacte à une question formulée en langue naturelle en s'appuyant sur les connaissances contenues dans des graphes comme DBpedia ou Wikidata. La tâche de KGQA nécessite donc à la fois de comprendre la question formulée en langage naturel et d'extraire l'information pertinente d'un graphe de connaissances pour y répondre.

Des approches symboliques ont été proposées. Elles traduisent les questions en langage naturel en requêtes SPARQL via des patrons, des règles linguistiques ou des modèles de traduction syntaxique, e.g. (Unger et al., 2012). Si ces approches garantissent précision et traçabilité, elles sont sensibles à la variabilité du langage et difficilement adaptables à grande échelle. Des travaux plus récents s'intéressent au développement de systèmes hybrides croisant LLMs et web sémantique (Pan et al., 2024). Des systèmes tels que QAnswer (Ruseti et al., 2015) et SPARQL-QA (Borroto et al., 2022) figurent parmi les approches KGQA les plus performantes, exploitant respectivement les lexicalisations de Wikipédia et la traduction automatique neuronale pour générer des requêtes SPARQL. Toutefois, des défis restent ouverts. En effet, si les LLMs facilitent la compréhension du langage, ils manquent de robustesse et de fiabilité. De

plus, les ambigüités sont encore mal gérées et les interactions se limitent souvent à un seul prompt.

Dans cet article, nous proposons l'approche AlasQA, un système neurosymbolique répondant à des questions posées en langue naturelle en combinant la fiabilité des graphes de connaissances par l'utilisation d'un langage formel comme SPARQL et la puissance des LLMs. AlasQA s'appuie sur Sparklis (Ferré, 2016) comme outil intermédiaire de construction interactive de requêtes SPARQL garantissant la validité des requêtes produites à partir du LLM. L'utilisateur pose une question en langue naturelle, le LLM génère une ou plusieurs séquences de commandes, qui permettent via Sparklis de construire itérativement une requête SPARQL traçable et ancrée dans les données. Le guidage interactif de Sparklis évite les résultats vides, tandis que le contexte apporté par l'enchaînement des commandes dans une séquence facilite la désambiguïsation des homonymes. Le système propose ensuite un retour en langue naturelle basé sur les résultats de la requête. L'utilisateur peut alors affiner ou compléter sa requête soit en langue naturelle, soit en naviguant manuellement dans l'interface de Sparklis. Des expérimentations menées sur les jeux de données QALD et TEXT2SPARQL valident l'intérêt de cette approche hybride.

2 État de l'art

Les premières méthodes de KGQA reposent sur le Web sémantique et le TAL classique. L'objectif est de traduire une question en langage naturel en une requête formelle (SPARQL) exécutée sur un graphe RDF. Certaines approches exploitent le *parsing* sémantique, d'autres des patrons linguistiques (Unger et al., 2012) ou des interfaces interactives guidant l'utilisateur (Ferré, 2016). Elles garantissent une excellente traçabilité mais souffrent d'un manque de flexibilité linguistique et d'une portabilité limitée entre domaines (Affolter et al., 2019). Une autre grande famille d'approches concerne l'utilisation de LLMs. Cela permet de répondre directement aux questions ou de générer des requêtes SPARQL pour interroger le graphe (Padonou et al., 2024). La force des LLMs réside dans leur couverture linguistique et leur robustesse. Néanmoins, leurs connaissances internes sont figées, coûteuses à mettre à jour et difficiles à vérifier (Thorne et al., 2018; Zhuang et al., 2023). Ils restent fragiles face aux opérateurs symboliques complexes, malgré les progrès liés au *fine-tuning* et à l'ingénierie de *prompts* (Wei et al., 2022). Afin de dépasser ces limites, certains proposent des approches hybrides avec des méthodes du web sémantique (Pan et al., 2024). Par exemple, les approches RAG (*Retrieval Augmented Generation*) (Lewis et al., 2020) injectent du contexte provenant des graphes dans le prompt du LLM. D'autres approches confient au LLM le rôle d'orchestrateur d'outils, capable d'appeler un moteur SPARQL ou une API pour raisonner (Yao et al., 2023; Lu et al., 2023). Ces méthodes conjuguent la souplesse des LLMs et la fiabilité des graphes de connaissances, même si actuellement des limites persistent comme la difficulté de généralisation entre graphes, la gestion limitée des ambiguïtés, ou la faible robustesse sur les opérateurs complexes. Notre proposition, AlasQA, s'inscrit dans cette dynamique hybride : elle combine la rigueur des graphes de connaissances avec la flexibilité des LLMs, grâce à un outil interactif de construction de requêtes SPARQL garantissant traçabilité et transparence.

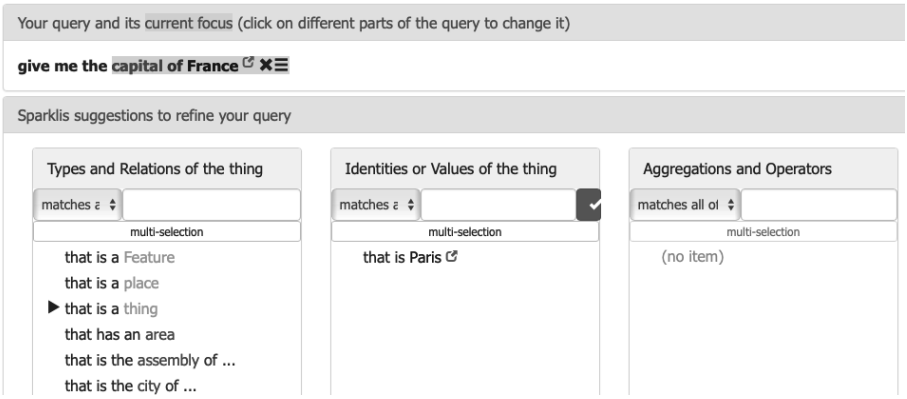

FIG. 1 – *Interface Sparklis avec la question "Quelle est la capitale de la France ?"*

3 Préliminaires : Sparklis

Sparklis est un outil interactif d'interrogation de données du web sémantique représentées sous forme de graphes de connaissances en RDF. Il repose sur la création incrémentale de requêtes SPARQL par un utilisateur final, sans nécessiter de connaissance de la syntaxe SPARQL ni du vocabulaire du graphe. L'utilisation la plus courante de Sparklis se fait via son interface graphique, mais une API est aussi disponible. La figure 1 illustre l'interface utilisateur de Sparklis lors d'une requête sur le graphe DBpedia [1] demandant la capitale de la France.

Dans la suite de l'article, nous utilisons les notions du web sémantique suivantes :

— **Termes RDF** : nœuds du graphe RDF, incluant les URIs, les littéraux (typés ou non typés) et les nœuds anonymes (e.g., France, Paris) ;
— **Variables SPARQL** : éléments d'un ensemble infini distinct de l'ensemble des termes RDF, utilisés pour représenter des valeurs inconnues dans une requête ;
— **Classes / types RDF** : catégories d'entités (e.g., `Country`) qui permettent de filtrer les résultats selon leur nature ;
— **Propriétés / relations RDF** : types des liens entre termes (e.g., `height`, `capital`), explorables dans le sens sujet → objet ou objet → sujet.

Nous utilisons aussi des notions propres à Sparklis liées à la navigation :

— **Focus** : position courante dans la requête, sur laquelle les actions de navigation s'appliquent (surligné en vert dans la figure 1). Le focus peut être déplacé dans l'arbre syntaxique de la requête ;
— **Suggestions** : éléments proposés dynamiquement pour insertion dans la requête au focus courant. Il peut s'agir d'entités, de classes, de propriétés ou d'opérations (par exemple, agrégation ou tri). Les suggestions sont adaptées au contexte du focus et reflètent les données disponibles dans le graphe RDF interrogé ;
— **Contraintes** : filtres applicables sur les suggestions (e.g., `matches "Einstein"`).

1. `https://fr.dbpedia.org`

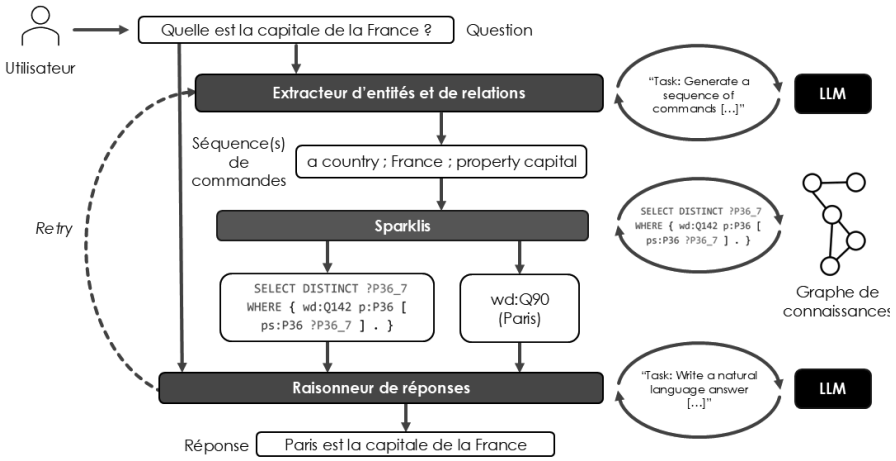

FIG. 2 – *Flux d'informations d'AlasQA.*

4 AlasQA : système neurosymbolique de KGQA

Dans cette section, nous présentons AlasQA. Nous commençons par une vue d'ensemble (section 4.1), puis définissons le jeu de commandes (section 4.2). Nous expliquons ensuite comment AlasQA explore l'espace des suggestions de Sparklis pour choisir les plus pertinentes pour une séquence de commandes (section 4.3) et proposons différentes stratégies de génération de séquences de commandes (section 4.4).

4.1 Approche générale

La figure 2 illustre la manière dont AlasQA transforme une question en langue naturelle en une requête SPARQL, puis restitue la réponse correspondante. Le processus s'effectue en trois étapes principales, articulées autour des interactions entre le LLM et Sparklis, via une séquence de commandes intermédiaires.

(1) À partir de la question en langue naturelle, les entités et les relations sont extraites à l'aide d'un LLM produisant une séquence de commandes pour Sparklis. Dans l'exemple de la figure, la séquence est `<a country ; France ; property capital>`.

(2) Cette séquence de commandes est ensuite fournie à Sparklis. AlasQA traite séquentiellement les commandes de la séquence, comme si un utilisateur interagissait avec Sparklis, afin de construire itérativement une requête SPARQL correcte et interroger le graphe de connaissances. Dans la figure 2, la requête SPARQL construite est `SELECT DISTINCT ?P36_7 WHERE{wd:Q142 p:P36[ps:P36 ?P36_7].}` et la réponse renvoyée par le graphe est `wd:Q90` qui correspond à l'URI de la ville de Paris dans le graphe de connaissances Wikidata.

(3) Enfin, la requête SPARQL et la réponse sont fournies à un raisonneur permettant de remodeler ou d'enrichir la requête obtenue. Dans cette dernière étape, AlasQA se sert d'un LLM pour traduire la réponse en langue naturelle.

4.2 Définition du langage des commandes

Une séquence de commandes se compose d'un ensemble de commandes séparées par des points-virgules et s'exécutant successivement (par exemple : `<France ; property capital>`). Ces commandes font référence à diverses actions, principalement des appels à l'API de Sparklis. Elles remplacent ainsi la navigation par l'interface et servent d'intermédiaire entre le LLM et Sparklis pour automatiser les interactions.

Le jeu de commandes se compose d'instructions inspirées de SPARQL et Sparklis :

— `a [`*class*`]` : recherche des entités d'une classe donnée (ex. : `a country`);
— `[`*entity*`]` : sélection directe d'une entité (ex. : `France`);
— `property [`*property*`]` : recherche d'une relation existante (sens direct ou inverse) pour un sous-ensemble des entités du focus courant (ex. : `property capital`).

Des commandes permettent de faire des comparaisons entre valeurs numériques ou dates :

— `higherThan [`*number*`]`, `lowerThan [`*number*`]`
(e.g., `<property weight ; higherThan 10>`);
— `after [`*date*`]`, `before [`*date*`]`
(e.g., `<property release date ; after 2000>`).

Enfin, d'autres commandes permettent de gérer des agrégations, des tris ou encore de limiter le nombre de résultats :

— `groupBy count` : groupement sur la dernière propriété explorée et comptage (e.g., `<a movie ; property film director ; groupBy count>`);
— `asc`, `desc` : tri croissant/décroissant des résultats;
— `limit [`*number*`]` : limite le nombre N de résultats;
— `offset [`*number*`]` : ignore les N premiers résultats.

4.3 Exploration de l'espace des suggestions

Dans cette section, nous expliquons comment, pour une commande donnée, une suggestion de Sparklis est sélectionnée (section 4.3.1) et présentons ensuite différentes stratégies d'exploration de l'espace des suggestions (section 4.3.2).

4.3.1 Traitement d'une commande

Lorsque AlasQA traite une commande, Sparklis propose un ensemble de suggestions à partir de contraintes relatives au focus actuel et des mots-clés fournis en paramètre de la commande. Ces suggestions peuvent être de différentes natures (relation, classe, terme). Parmi ces suggestions AlasQA sélectionne celle qui semble la plus cohérente avec la commande. On distingue trois cas en fonction de la nature de la commande traitée.

1. Si la commande est une entité, une classe ou une propriété, un appel à Sparklis retourne la liste des suggestions compatibles avec le focus courant et la commande.
2. Si la commande est une contrainte, une agrégation ou un tri, elle peut être convertie directement en une suggestion Sparklis.
3. Si la commande est `limit` ou `offset`, un post-traitement de la requête SPARQL est enregistré pour être appliqué à la fin.

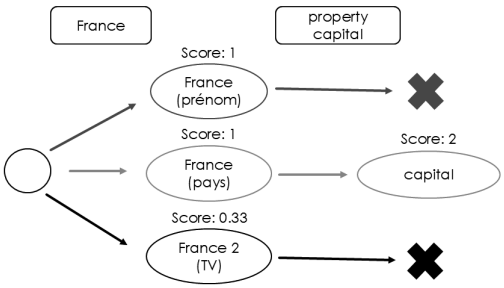

FIG. 3 – *Exemple de chemins d'exploration pour "`France ; property capital`".*

Pour le premier cas (entité, classe ou propriété), nous définissons un score pour guider le choix des suggestions les plus pertinentes. Ce score combine deux critères :

— *dist* : la proximité lexicale, mesurée par la distance de Levenshtein entre la chaîne en paramètre et le libellé de la suggestion ;

— *freq* : la pertinence statistique, mesurée par la fréquence d'apparition de la suggestion dans le graphe parmi les entités au focus courant. Par exemple, si le focus courant porte sur des *personnes mariées* et qu'une des suggestions est la propriété `child`. La pertinence de `child` pour ce focus est mesurée en calculant le nombre de personnes mariées dans le graphe qui possèdent la propriété `child`. Plus cette valeur est élevée, plus la suggestion apparaît comme pertinente.

Le score d'une suggestion pour une commande est défini par la formule suivante :

$$\text{score(cmd, sugg)} = \mathit{freq}(\mathit{sugg}, \mathit{focus}) \times \frac{1}{1 + \mathit{dist}(\mathit{cmd}, \mathit{sugg})}.$$

En cas d'égalité de score, un second critère permet de départager les suggestions : le nombre d'apparitions de la suggestion dans le graphe global et non plus seulement sur les éléments du focus. Cette heuristique part du principe qu'une suggestion plus fréquente dans le graphe est généralement plus pertinente. Il s'agit toutefois d'un choix local qui ne prend en compte que la commande courante. Or, une suggestion localement bonne pour une commande peut s'avérer bloquante pour les commandes suivantes.

4.3.2 Exploration de l'espace des suggestions pour une séquence de commandes

Nous avons défini trois tactiques d'exploration de l'espace des suggestions : gloutonne, exhaustive et par faisceau. Nous détaillons chacune dans la suite. Chaque séquence de suggestions est appelée un chemin d'exploration. La figure 3 illustre, pour la séquence `<France ; property capital>`, les chemins d'exploration possibles. Chaque nœud correspond à un état de Sparklis, avec un score cumulé et un ensemble de commandes restantes. Les transitions entre états résultent de l'application d'une suggestion donnée.

Tactique gloutonne La tactique gloutonne traite chaque commande de manière indépendante. Pour un état donné de Sparklis, elle applique toujours la meilleure suggestion locale, sans considérer l'impact sur les commandes suivantes. Cette approche a l'avantage d'être rapide mais peut aboutir à une solution globalement sous-optimale si une suggestion sélectionnée

localement bloque les commandes suivantes. Dans la figure 3, cette tactique échoue car elle suit le chemin rouge qui sélectionne le prénom France et qui n'a pas de propriété `capital`.

Tactique exhaustive Cette tactique a été proposée pour palier au problème des optimum locaux. Pour une séquence de commandes, au lieu de n'explorer qu'un seul chemin, plusieurs chemins sont testés. À chaque chemin est associé un score qui correspond à la somme des scores de chaque suggestion du chemin. L'objectif est d'identifier le chemin qui maximise le score. Afin de limiter la combinatoire, pour chaque commande, seules les n meilleures suggestions locales (selon leur score) sont conservées. Ensuite toutes les combinaisons possibles de suggestions sont explorées. Cette tactique identifie la meilleure séquence globale mais est la plus coûteuse en temps de calcul. Dans l'exemple de la figure 3, cette tactique teste les trois chemins (rouge, vert et noir) et trouve la solution.

Tactique par faisceau La dernière tactique s'inspire de la tactique exhaustive en explorant plusieurs chemins tout en limitant chaque commande aux n meilleures suggestions. Elle n'explore cependant pas toutes les combinaisons possibles de suggestions : parmi tous les chemins en cours de construction, seuls les k meilleurs (le faisceau) sont retenus pour l'étape suivante. Cette approche équilibre qualité de la solution et coût de calcul. Dans l'exemple de la figure 3, si on suppose $k = 2$, elle explore seulement les chemins rouge et vert.

4.4 Stratégies de génération de séquences de commandes

Différentes stratégies peuvent être utilisées pour définir la façon dont le LLM est utilisé par AlasQA afin de générer la séquence de commandes. Nous avons défini deux stratégies : *One-shot* (OS), qui génère une unique séquence de commandes et *Retry*, qui produit plusieurs séquences pour une même question et compare leurs résultats. De plus, nous proposons un traitement spécifique pour les questions dont la réponse est de type booléen.

One-shot (OS) Cette stratégie effectue un appel unique au LLM pour générer une séquence complète de commandes à partir d'un *prompt* donné. Cette séquence est ensuite exécutée dans Sparklis, comme vu dans la section précédente. Il s'agit de la stratégie la plus directe et la plus rapide à exécuter, car elle ne nécessite ni itération, ni interaction intermédiaire, ni mécanisme de correction par le LLM. Cependant, son efficacité dépend fortement de la qualité du *prompt*, notamment des commandes explicitement introduites dans le *prompt* comme par exemple les commandes d'ordonnancement (`asc`, `desc`); ainsi que des capacités du LLM à générer des commandes valides, cohérentes et pertinentes.

Retry La stratégie repose sur une idée simple : relancer l'appel au LLM tant que les résultats ne sont pas satisfaisants, privilégiant la fiabilité au détriment du temps de calcul. L'algorithme répète la stratégie *One-shot* jusqu'à obtenir un certain nombre de réponses identiques non vides, partant de l'hypothèse qu'une réponse vide correspond généralement à une mauvaise interprétation de la question plutôt qu'à un cas où l'absence de réponse est correcte. Cette hypothèse n'est pas toujours valide dans des cas réels car elle empêche la possibilité d'avoir des réponses vides légitimes. Une réponse non vide ne suffit pas, car le LLM génère fréquemment des séquences renvoyant des résultats sans pour autant répondre correctement à la question en raison d'hallucinations, d'erreurs de terminologie, ou de mauvaises interprétations. Nous faisons l'hypothèse qu'une bonne réponse, si elle est atteignable, a plus de chances de réap-

paraître de façon récurrente parmi les réponses générées. Une réponse n'est donc considérée comme valide que si elle est produite un certain nombre de fois (trois dans nos expériences).

Boolean Les stratégies précédentes ne permettent pas de traiter efficacement les questions dont la réponse est un booléen, car Sparklis ne génère que des requêtes SPARQL produisant des résultats non vides. Or, obtenir une réponse booléenne négative (par exemple, montrer qu'une personne n'est pas décédée) nécessite de raisonner sur l'absence de lien (dans l'exemple, l'absence d'une date de décès). Pour pallier cette limite, nous introduisons une stratégie dédiée à ce type de questions, reposant sur trois éléments : (1) un *prompt* spécialisé, formulé pour expliciter qu'entre deux entités une comparaison ou une vérification d'existence d'une propriété est attendue ; (2) la génération non pas d'une seule séquence de commandes, mais de deux séquences, reliées par un opérateur de comparaison (=, !=, <, >) ou par une comparaison d'ensembles (pour vérifier si des ensembles se recouvrent, sont disjoints ou vides) ; (3) l'introduction d'une commande spécifique `match [`*string*`]`, permettant d'effectuer une correspondance textuelle et de retourner une liste de suggestions sans imposer une sélection unique. Par exemple, pour traiter la question "Paris est-elle la capitale de la France ?", le LLM génère deux séquences : <`France ; property capital`> et <`match Paris`>. Celles-ci sont reliées par l'opérateur =, exprimant que la valeur attendue pour la propriété `capital` doit correspondre à `Paris`. Si la comparaison est vraie, la réponse est `True`, sinon `False`.

5 Évaluations expérimentales

Cette section présente les expériences dont le code et les données sont disponibles en ligne [2]. Tout d'abord, nous présentons les jeux de données (section 5.1). Ensuite nous comparons les différentes tactiques d'exploration des suggestions (section 5.2) et stratégies de génération de séquences de commandes (section 5.3). Enfin, nous discutons de l'impact du modèle LLM choisi et comparons nos résultats à ceux de systèmes existants (section 5.4).

Chaque expérience est exécutée trois fois afin d'assurer la fiabilité des résultats. Leur évaluation repose sur des mesures standards : précision, rappel et F1-score macro. Contrairement à certaines approches attribuant une précision de 1 à une réponse vide, nous considérons qu'une telle réponse constitue une erreur et lui attribuons une précision de 0. Les premières évaluations présentées ci-dessous utilisent le modèle `mistral-nemo-instruct-2407@q4_k_m`. Des évaluations menées avec d'autres LLMs sont présentées en section 5.4. De plus, la taille du faisceau pour les approches par faisceau est de 3.

5.1 Jeux de données

QALD-9-plus et QALD-10 Ces jeux de données associent des questions en langage naturel, disponibles en plusieurs langues, à des requêtes SPARQL. QALD-9-Plus (Perevalov et al., 2022) couvre les graphes DBpedia et Wikidata, avec des requêtes Wikidata adaptées depuis DBpedia. QALD-10 (Usbeck et al., 2024) se concentre uniquement sur Wikidata. Les questions couvrent plusieurs catégories (factuelles, quantitatives, superlatives, comparatives, agrégatives), et les réponses attendues peuvent être des entités (URI), des littéraux ou des booléens. Pour l'évaluation, nous utilisons une vérité terrain réévaluée, i.e., les requêtes

2. `https://github.com/BaptisteAmice/AlasQA`

Catégorie	Dataset	Nb questions d'origine	Nb questions utilisées	Nb questions modifiées
QALD-9-Plus	DBpedia Train	408	341	0
	DBpedia Test	150	115	3
	Wikidata Train	371	353	11
	Wikidata Test	136	126	4
QALD-10	Wikidata	394	387	5

TAB. 1 – *Questions de QALD-9-Plus et QALD-10.*

Dataset	Type	OS-GREEDY			OS-BEAM			OS-DFS		
		Précision	Rappel	F1	Précision	Rappel	F1	Précision	Rappel	F1
QALD-9-Plus Wikidata train + test (EN)	Tous	29 ± 1	29 ± 1	27 ± 1	30 ± 1	30 ± 2	28 ± 1	31 ± 0	31 ± 0	$\mathbf{29 \pm 0}$
	bool	10 ± 7	10 ± 7	10 ± 7	6 ± 1	6 ± 1	6 ± 1	9 ± 1	9 ± 1	9 ± 1
	uris	29 ± 1	29 ± 1	26 ± 0	31 ± 2	31 ± 3	28 ± 2	31 ± 1	32 ± 1	28 ± 1
	literals	36 ± 3	37 ± 4	36 ± 4	37 ± 1	38 ± 2	37 ± 2	38 ± 3	39 ± 3	38 ± 3
QALD-10 Wikidata (EN)	Tous	19 ± 2	22 ± 2	19 ± 2	19 ± 1	22 ± 1	20 ± 1	20 ± 1	22 ± 1	$\mathbf{20 \pm 1}$
	bool	3 ± 2	3 ± 2	3 ± 2	2 ± 1	2 ± 1	2 ± 1	1 ± 1	1 ± 1	1 ± 1
	uris	29 ± 4	33 ± 4	29 ± 4	28 ± 2	32 ± 2	28 ± 2	30 ± 1	33 ± 2	30 ± 1
	literals	14 ± 0	17 ± 0	15 ± 1	17 ± 2	20 ± 2	17 ± 2	15 ± 1	19 ± 1	16 ± 1
TEXT2SPARQL Corporate	Tous	13 ± 3	31 ± 1	11 ± 3	12 ± 1	27 ± 4	11 ± 2	13 ± 1	29 ± 5	$\mathbf{12 \pm 1}$
	bool	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	uris	15 ± 5	41 ± 2	14 ± 4	14 ± 1	32 ± 5	13 ± 3	14 ± 1	32 ± 5	14 ± 1
	literals	9 ± 2	14 ± 0	7 ± 3	11 ± 1	20 ± 7	11 ± 2	14 ± 4	30 ± 6	12 ± 3

TAB. 2 – *Performances en pourcentages avec* ***mistral-nemo-instruct-2407@q4_k_m*** *de la stratégie One-Shot selon trois tactiques : gloutonne, par faisceau et exhaustive.*

SPARQL ont été réexécutées sur les versions actuelles des graphes. Enfin, les éléments obsolètes ou invalides des jeux de données ont été corrigés ou supprimés, comme résumé en table 1. Dans la suite nous présentons les résultats concernant les questions en anglais. Toutefois, nous avons aussi mené des expériences en français. Les résultats sont inférieurs à l'anglais (en moyenne 5% de différence pour le F1-score). Cela est du à une phase de traduction car les termes utilisés dans les graphes et donc dans Sparklis sont en anglais.

TEXT2SPARQL Nous avons également participé au *First International TEXT2SPARQL Challenge 2025*[3]. Le jeu de données utilisé est divisé en deux parties : 100 questions portant sur *DBpedia* (avec des versions en anglais et en espagnol) et 50 questions sur un graphe dit *corporate* représentant un environnement professionnel fictif.

5.2 Comparaison des tactiques d'exploration de l'espace des suggestions

Nous comparons trois tactiques d'exploration de l'espace des suggestions : gloutonne (OS-GREEDY), exhaustive (OS-DFS) et par faisceau (OS-BEAM). Nous utilisons la stratégie *One-Shot* (OS). Les résultats sont donnés dans la table 2. Les performances observées des tactiques exhaustive (OS-DFS) et par faisceau (OS-BEAM) sont légèrement supérieures à celles de la tactique gloutonne (OS-GREEDY). Ce résultat confirme l'intérêt de considérer une séquence complète de commandes plutôt que chaque commande individuellement. Comme attendu, OS-

3. `https://text2sparql.aksw.org/`

Dataset	Type	OS-BEAM			RETRY-BEAM			BOOLEAN-RETRY-BEAM
		Précision	Rappel	F1-score	Précision	Rappel	F1-score	F1-score
QALD-9-Plus Wikidata train + test (EN)	Tous	30 ± 1	30 ± 2	28 ± 1	41 ± 1	42 ± 1	$\mathbf{39 \pm 1}$	-
	bool	6 ± 1	6 ± 1	6 ± 1	0 ± 0	0 ± 0	0 ± 0	60 ± 4
	URIs	31 ± 2	31 ± 3	28 ± 2	43 ± 1	44 ± 1	40 ± 1	-
	Literals	37 ± 1	38 ± 2	37 ± 2	51 ± 1	53 ± 2	51 ± 1	-
QALD-10 (EN)	Tous	19 ± 1	22 ± 1	20 ± 1	25 ± 0	29 ± 1	$\mathbf{26 \pm 0}$	-
	bool	2 ± 1	2 ± 1	2 ± 1	2 ± 3	2 ± 3	2 ± 3	38 ± 7
	URIs	28 ± 2	32 ± 2	28 ± 2	38 ± 1	42 ± 1	38 ± 1	-
	Literals	17 ± 2	20 ± 2	17 ± 2	21 ± 2	26 ± 1	22 ± 2	-

TAB. 3 – *Performances en pourcentages avec* ***mistral-nemo-instruct-2407@q4_k_m*** *de la stratégie One-Shot comparées aux stratégies Retry et Boolean.*

DFS fournit de meilleurs résultats qu'OS-BEAM, tandis que ce dernier présente un avantage en termes de temps d'exécution. Par exemple, pour QALD-10, OS-GREEDY s'exécute en moyenne en 41s, OS-BEAM en 94s et OS-DFS en 201s. On note aussi que les scores obtenus sur QALD-10 sont inférieurs à ceux obtenus sur QALD-9-Plus. En particulier les questions avec des réponses de type littéral sont moins bien traitées que celles ciblant des URIs, alors que la tendance est inversée sur QALD-9-Plus. Cela pourrait indiquer que les questions avec des réponses de type littéral dans QALD-10 impliquent des traitements plus complexes (calculs, conditions numériques) difficiles à modéliser avec des chaînes de commandes simples.

Sur le challenge TEXT2SPARQL, notre approche a obtenu un F1-score moyen de 13%, nous classant 7ème parmi les 12 équipes. Notons que l'approche n'était pas finalisée lors de notre participation et utilisait une variante de la stratégie *Retry* limitée dans le temps. Nous sommes aujourd'hui capables d'obtenir un F1-score de 12% sur le jeu corporate seulement avec la stratégie *One-Shot*, là où nous obtenions 13% à l'aide d'une stratégie plus gourmande.

5.3 Comparaison des stratégies de génération de commandes

Nous avons comparé les stratégies de génération de séquences de commandes : *One-Shot* (OS-BEAM) et *Retry* (RETRY-BEAM), ainsi que l'optimisation pour les questions dont la réponse attendue est booléenne (BOOLEAN-RETRY-BEAM). Les résultats sont présentés à la table 3. Par souci de concision, seule la tactique par faisceau est représentée, les conclusions ne variant pas selon la tactique utilisée.

On constate que la stratégie *Retry* obtient des scores significativement supérieurs à ceux de la stratégie *One-Shot*. Ainsi, pour QALD-9-Plus avec `Wikidata train + test (EN)`, le F1-score global passe de 28 % en *One-shot* à 39 % en *RETRY-BEAM* (soit +11 %).

En revanche, pour les questions dont la réponse est booléenne, les deux stratégies précédentes sont peu efficaces. Il apparaît plus pertinent de détecter ce type de questions (les LLM sont capables de les distinguer des questions factuelles avec un score de détection de 98,5%), puis d'appliquer l'approche dédiée, qui fournit des résultats nettement supérieurs, comme le montre la dernière colonne de la table 3.

5.4 Discussions

Impact du choix du LLM sur les performances Nous avons évalué quatre LLMs sur le jeu de données QALD-9-Plus Wikidata test (EN) avec la tactique par faisceau : trois modèles de

taille raisonnable en stratégie *Retry* (`qwen2.5-7b-instruct-1m` (`7B`), `mistralnemo` et `gpt-4o mini`) et un modèle plus grand (`gpt-4o`) en stratégie *One-shot*. On constate des F1-scores similaires entre les modèles : `qwen2.5` obtient un score de 35%, `mistral-nemo` de 36%, `gpt-4o mini`) de 36% et `gpt-4o` 36%. Ainsi, un LLM plus puissant comme `gpt-4o` est capable, en un seul essai avec la stratégie *One-shot*, d'obtenir des performances comparables à celles de modèles plus petits utilisant la stratégie *Retry*.

Comparaison avec d'autres systèmes de KGQA Dans le papier de présentation de QALD-9-Plus, on constate que QAnswer (Ruseti et al., 2015), le système le plus performant sur ce *benchmark*, obtient un F1-score de 41% sur la portion Wikidata du jeu de données. AlasQA avec la stratégie *Retry* et l'optimisation pour les questions dont la réponse est un booléen obtient un F1-score de 44%. Nos résultats dépassent donc légèrement ceux de QAnswer sur ce jeu de données, et ce avec un modèle de taille modérée (Mistral).

Dans le papier de présentation de QALD10, en revanche, le système QAnswer obtient un F1-score de 58%, se classant deuxième derrière le système SPARQL-QA (Borroto et al., 2022) qui obtient un F1-score de 60% alors qu'AlasQA obtient un F1-score de 30%. Nous pensons que dans QALD10 certaines catégories de questions, comme les comparaisons, restent peu ou mal traitées par AlasQA et nécessiteraient des stratégies plus adaptées. Toutefois, notons que AlasQA a un coût plus faible que ces modèles. En effet, il ne nécessite pas de *fine-tuning* contrairement à SPARQL-QA et est moins sensible à la langue, contrairement à QAnswer.

6 Conclusion

Nous avons présenté AlasQA, une nouvelle approche neurosymbolique LLM-Sparklis de l'interrogation en langue naturelle de graphes de connaissances. Les LLMs permettent d'absorber la variabilité de la langue naturelle et de traduire une question en une suite de commandes interprétables par Sparklis. Sparklis garantit la validité des requêtes SPARQL construites grâce à un ancrage dans les données. Ses suggestions définissent un espace de recherche ouvert à différentes tactiques et stratégies de recherche. Nous obtenons des résultats encourageants sur des jeux de questions de référence, QALD et TEXT2SPARQL, sans aucune spécialisation aux données cibles ni *fine-tuning* des LLMs. Une perspectives d'amélioration est la mise en place d'un mécanisme similaire à celui permettant de traiter les questions booléennes, mais dédié aux questions nécessitant des comparaisons entre sous-requêtes, comme les questions de localisation temporelle et spatiale (e.g., "Who became president after JFK died ?").

Références

Affolter, K., K. Stockinger, et A. Bernstein (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal 28*(5), 793–819.

Borroto, M., F. Ricca, B. Cuteri, et V. Barbara (2022). Sparql-qa enters the qald challenge. In *Proc. of the 7th Natural Language Interfaces for the Web of Data (NLIWoD) co-located with ESWC*, Volume 3196, pp. 25–31.

Ferré, S. (2016). Sparklis : An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web 8*(3), 405–418, doi: 10.3233/SW-150208. hal-01485093.

Lewis, P., E. Perez, A. Piktus, et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Volume 33, pp. 9459–9474. Curran Associates, Inc.

Lu, P., B. Peng, H. Cheng, et al. (2023). Chameleon : Plug-and-Play Compositional Reasoning with Large Language Models. *Advances in Neural Information Processing Systems 36*.

Padonou, A. E., P. Cellier, et S. Ferré (2024). Étude de l'utilisation des modèles de langages pour l'interrogation en langue naturelle des graphes de connaissances. In *EGC'24*.

Pan, S., L. Luo, Y. Wang, et al. (2024). Unifying Large Language Models and Knowledge Graphs : A Roadmap. *IEEE TKDE 36*(7), 3580–3599.

Perevalov, A., D. Diefenbach, R. Usbeck, et al. (2022). QALD-9-plus : A Multilingual Dataset for Question Answering over DBpedia and Wikidata Translated by Native Speakers. In *2022 IEEE 16th Int. Conf. on Semantic Computing*, pp. 229–234. ISSN : 2325-6516.

Ruseti, S., A. Mirea, T. Rebedea, et S. Trausan-Matu (2015). Qanswer-enhanced entity matching for question answering over linked data. In *CLEF (Working Notes)*, pp. 28–35.

Thorne, J., A. Vlachos, C. Christodoulopoulos, et al. (2018). FEVER : a Large-scale Dataset for Fact Extraction and VERification. In *Conf. of the North American Chapter of the ACL : Human Language Technologies, Volume 1*, pp. 809–819. ACL.

Unger, C., L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, et P. Cimiano (2012). Template-based Question Answering over RDF data. In *Proceedings of the 21st international conference on World Wide Web*, pp. 639–648.

Usbeck, R., X. Yan, A. Perevalov, L. Jiang, J. Schulz, A. Kraft, C. Möller, J. Huang, J. Reineke, A.-C. N. Ngomo, M. Saleem, et A. Both (2024). Qald-10 – the 10th challenge on question answering over linked data : Shifting from dbpedia to wikidata as a kg for kgqa. *Semantic Web 15*(6), 2193–2207.

Wei, J., X. Wang, D. Schuurmans, et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems 35*.

Yao, S., J. Zhao, D. Yu, et al. (2023). ReAct : Synergizing Reasoning and Acting in Language Models. In *Int. Conf. on Learning Representations (ICLR)*.

Zhuang, Y., Y. Yu, K. Wang, et al. (2023). ToolQA : A Dataset for LLM Question Answering with External Tools. *Advances in Neural Information Processing Systems 36*, 50117–50143.

Summary

We focus on the task of querying a knowledge graph (KG) using natural language. KG can be queried reliably through formal languages such as SPARQL; however, this requires a complex translation from the natural language to the formal language. Large Language Models (LLMs) are able to directly answer natural language questions but provide no guarantees regarding the validity of the answers they generate. We propose a neurosymbolic system, called AlasQA, which answers natural language questions by combining the reliability of a formal language like SPARQL with the power of LLMs. The approach relies on an intermediate tool for interactive SPARQL query construction. Experiments conducted on the QALD and TEXT2SPARQL datasets validate the relevance of this hybrid method.