Adaptations des modèles IA pour interroger en langage naturel la base de données LandMatrix

Fatiha Ait Kbir[****,***], Jérémy Bourgoin[*,***♯], Rémy Decoupes[**,***], Marie Gradeler[♯,***], Roberto Interdonato[*,***],

[**]INRAE, F-34398 Montpellier, France
prenom.nom@inrae.fr,
[*]CIRAD, F-34398 Montpellier, France
prenom.nom@cirad.fr
[***]TETIS, Univ. Montpellier, AgroParisTech, CIRAD, CNRS, INRAE, Montpellier 34090, France
[****]Stagiaire au CIRAD, F-34398 Montpellier, France
fatihaaitkbir.contact@gmail.com
[♯]International Land Coalition, Rome, Italy
m.gradeler@ifad.org

Résumé. L'initiative Land Matrix vise à fournir des données fiables sur les acquisitions de terres à grande échelle dans les pays à revenus faibles ou intermédiaires. Bien que ces données soient reconnues dans le monde académique, elles restent peu utilisées dans l'action publique, en grande partie en raison de la complexité d'accès et d'exploitation, nécessitant une expertise informatique et une bonne connaissance du schéma de la base de données.

L'objectif de ce travail est de comparer plusieurs optimisations de *Large Language Models* (Prompt Engineering, RAG, Agents) pour interroger différents systèmes de base de données (GraphQL et requêtes REST). Les expérimentations sont reproductibles et une démonstration est accessible en ligne : `https://github.com/tetis-nlp/landmatrix-graphql-python`.

1 Introduction

L'Initiative Land Matrix [1] (LMI) est une initiative indépendante de suivi des terres qui collecte et cherche à vérifier des informations sur les acquisitions de terres à grande échelle. Elle conserve les informations de transactions impliquant un transfert de droits d'utilisation, de contrôle ou de propriété des terres par vente, location ou concession couvrant 200 hectares (ha) ou plus, conclues depuis l'année 2000 (Anseeuw et al., 2012).

La base de données de LMI propose deux interfaces de programmation applicative (API), REST et GraphQL, qui peuvent être utilisées pour récupérer les données.

Bien qu'elles soient une référence mondiale sur les phénomènes d'acquisition de terres dans la communauté académique, les données de la LMI restent sous-utilisées pour contribuer

1. https ://landmatrix.org

à l'action publique, et ce malgré des efforts considérables pour en améliorer l'accessibilité. En effet, accéder et utiliser efficacement ces données peut être complexe, car cela nécessite une compréhension approfondie à la fois du jeu de données mais aussi des interfaces de requête pour extraire les données pertinentes et exploitables.

L'objectif principal de ce travail est de simplifier l'accès à la base de données LMI. Pour cela, nous avons adapté des approches Text-to-SQL (Hong et al., 2024) afin de guider les modèles de langage (LLMs) à générer des requêtes REST et GraphQL à partir de questions formulées en langage naturel.

Bien que certaines méthodes de "*prompt engineering*" aient considérablement amélioré les performances du Text-to-SQL, elles restent insuffisantes pour relever les défis posés par la base de données LMI. Cela est particulièrement vrai lorsque les requêtes des utilisateurs sont ambiguës ou très complexes, ce qui peut limiter la capacité des modèles à identifier et traiter tous les éléments pertinents de la question. Par exemple, si un utilisateur recherche des transactions liées à un pays spécifique, le modèle peut parfois échouer à identifier correctement le pays et à le lier à son identifiant correspondant (*primary key*).

La contribution de cet article est de comparer différents modèles de langage avec diverses optimisations pour l'extraction de données de la Land Matrix à partir de questions en langage naturel. L'étude évalue trois modèles *open-weight* : Llama3-8B, Mixtral-8x7B-instruct et Codestral-22B, en utilisant un corpus de demandes réelles, en langage naturel, de requêtes provenant d'utilisateur non-informaticien et leurs requêtes sous forme d'API paramétrées par les experts REST et GraphQL de la LMI. Ce travail est reproductible et permet l'ajout de modèles supplémentaires. Trois techniques d'optimisation sont comparées : *prompts engineering*, la génération augmentée par recherche d'information (*RAG*) et les agents.

2 État de l'art

Les capacités génératives des modèles de langues offrent de nouvelles opportunités pour interagir facilement avec des systèmes de bases de données sans nécessairement connaître leur syntaxe. Depuis 2017 et la publication de *Attention is All You Need* (Vaswani et al., 2017), le traitement automatique de la langue (TAL) a connu une amélioration rapide de ses performances grâce aux *transformers*, une nouvelle architecture de réseaux de neurones. Deux types de modèles existent. Les premiers, basés sur des couches d'encodeurs, tels que BERT (Devlin et al., 2019), ne sont pas génératifs mais permettent de traiter différentes tâches classiques en TAL (reconnaissance d'entités nommées, classification de texte, analyse de sentiments, etc.). Le second type comporte des modèles de plus grande taille avec des capacités génératives comme GPT-3.5 d'OpenAI. Nous nous focalisons, dans cette étude, sur le deuxième type de modèles, les LLMs.

Ces assistants personnels possèdent différentes capacités de génération de texte. Ils peuvent répondre à des questions, résumer ou bien encore traduire vers d'autres langues (Zhao et al., 2023). Ils peuvent également générer des langages informatiques, comme des codes de programmation ou des requêtes de bases de données (Liu et al., 2024). D'ailleurs, selon l'étude de Zheng et al. (2023), les sujets les plus fréquemment abordés par les chatbots concernent la génération de code (débogage, résolution d'erreurs, génération de code).

Pour obtenir de meilleurs résultats et des requêtes de bases de données plus précises, plusieurs techniques peuvent être utilisées. La première, appelée *prompt engineering* (Brown

et al., 2020), consiste à formuler soigneusement les questions adressées au chatbot. La structure du *prompt* peut varier en fonction du modèle et de ses ré-entraînements. La deuxième option, le *few-shot learning* (Brown et al., 2020), consiste à fournir une série d'exemples dans le *prompt* pour aider le modèle à répondre plus efficacement. Une autre approche, un peu similaire, consiste à enrichir les *prompts* avec des connaissances externes issues d'une base de données à jour. Connue sous le nom de *Retrieval Augmented Generation* (RAG) (Lewis et al., 2021), cette méthode fournit des informations pertinentes pour aider le modèle à formuler sa requête (par exemple, une liste des valeurs possibles pour une colonne d'une table). D'autres méthodes incluent le ré-entraînement de modèle via le *fine-tuning* ou l'alignement (Balaguer et al., 2024), mais elles nécessitent des ressources matérielles importantes.

La recherche en Text-to-SQL est très active. Plusieurs études (Hong et al., 2024; Mohammadjafari et al., 2024) proposent des chronologies de différentes techniques (de celles basées sur des règles jusqu'à l'utilisation avancée de LLMs). Parmi ces techniques, l'approche RAG est souvent plébiscitée (Ziletti et D'Ambrosi, 2024). Dans notre travail, nous avons renforcé cette méthode en y intégrant une approche basée sur des agents pour guider encore davantage les modèles. Enfin, il apparaît que ce champs de recherche est en constante amélioration, aussi, nous proposons un jeu de données avec des couples de demandes formulées en langage naturel et leurs requêtes ainsi qu'un banc de comparaison évolutif basées sur un besoin réel, l'usage des données de la Land Matrix.

3 Méthodologies

L'objectif de ce travail est de comparer différents modèles de langage (LLMs) avec diverses optimisations pour extraire des données de la Land Matrix en utilisant des questions en langage naturel. Pour ce faire, différents partenaires de la Land Matrix ont rassemblé un ensemble d'environ 60 requêtes REST ou GraphQL issues de demandes réelles d'utilisateurs non technicien.

Dans cette étude, nous comparons trois modèles *open-weight* : Llama3-8B, un modèle *Mixture of Experts* ré-entraîné sur un jeu de données d'instructions (Mixtral-8x7B-instruct), et un LLM spécialisé dans la génération de code (Codestral-22B). Ce travail est reproductible, et des modèles supplémentaires peuvent être inclus.

Enfin, nous proposons trois optimisations. La première est le *prompt engineering*, la deuxième est la méthode RAG (*Retrieval Augmented Generation*), et la dernière implique l'utilisation d'agents LLM (*LLM Agents*).

3.1 Prompt Engineering

Pour générer des requêtes (GraphQL | REST), un *prompt* spécifique a été conçu. Celui-ci prend en compte la question de l'utilisateur, tout en fournissant un contexte détaillé incluant le schéma de la base de données. Ce schéma contient une description des attributs, leurs relations, ainsi qu'une liste des attributs possibles et de leurs valeurs correspondantes dans la base de données LMI. Par ailleurs, afin de guider le modèle, des exemples de paires question-requête sont intégrés (*few shots learning*). Des règles spécifiques et des informations relatives à l'API LMI sont également ajoutées. Le modèle est paramétré avec un rôle de spécialiste (GraphQL | REST).

3.2 Génération augmentée par recherche d'information (RAG)

Cela consiste à enrichir le précédent *prompt* avec des questions en langage naturel et leurs requêtes correspondantes, similaires à la question d'entrée. C'est une optimisation du *few shot learning*. Pour cela, la demande de l'utilisateur est encodée en *embedding* et comparée à l'ensemble de notre jeux de données préalablement encodé également avec le modèle all-mpnet-base-v2, un *sentence-transformers*. Les k questions les plus proches de la question de l'utilisateur, ainsi que leurs requêtes (GraphQL | REST) associées, sont sélectionnées et intégrées au *prompt*.

3.3 Agents

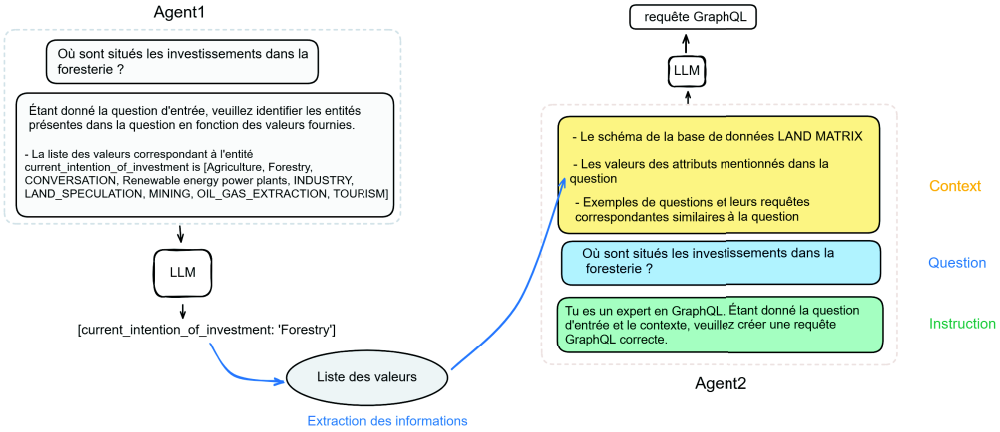

FIG. 1 – *Pipeline de l'optimisation par Agents : un autre modèle de langage est utilisé pour extraire les filtres demandés par l'utilisateur, et ses sorties sont ajoutées au prompt précédent.*

La collaboration multi-agents consiste à combiner plusieurs agents (des modèles spécialisés ou généralistes) pour diviser les tâches et s'échanger des informations. Dans notre cas, nous avons utilisé deux agents : le premier pour l'extraction d'entités et le second pour la génération de requêtes (voir Fig. 1).

Nous commençons par extraire les entités de la question de l'utilisateur à l'aide du premier agent. Ce dernier retourne des valeurs prédéfinies associées aux entités dans la base de données LMI sous forme de paires [entité : valeur]. Cette liste est ensuite utilisée pour récupérer toutes les valeurs pertinentes pour ces attributs à partir d'un ensemble prédéfini, enrichissant ainsi le contexte pour le second agent. Le second agent est responsable de la génération de la requête en intégrant les informations pertinentes concernant les attributs mentionnés dans la question. De plus, nous avons amélioré le processus en incluant des paires question-requête similaires grâce à la méthode RAG et en y ajoutant des informations pertinentes de la base de données.

Cette approche permet d'enrichir le *prompt* de génération de requêtes avec des informations contextuelles spécifiques, sans le surcharger avec toutes les variables possibles. Un trop grand nombre de variables dans le *prompt* peut en effet réduire la capacité du modèle à traiter les informations avec précision, entraînant ainsi des performances sous-optimales.

4 Expérimentations et résultats

Les différentes méthodes présentées sont évaluées à travers trois aspects. Le premier est la **vérification de la syntaxe**, il s'agît de s'assurer que les requêtes ne génèrent pas d'erreur lorsqu'elles sont soumises à l'API de LMI. Le second consiste à vérifier la bonne **détection des attributs de la base de données ainsi que leurs valeurs pour le filtrage**. Enfin, nous vérifions également **l'exactitude des données récupérées**, particulièrement lorsque les requêtes générées et celles des expertes sont différentes.

4.1 Validité syntaxique des requêtes générées

La table 1 présente les résultats concernant la validité des requêtes générées en REST ou GraphQL. Llama3 et Mixtral peinent à générer des requêtes valides pour l'API REST, seule l'approche Agent avec Codestral permet de générer des requêtes REST valides. En revanche, pour GraphQL, il semble que les LLMs soient plus à l'aise avec cette syntaxe. Il est également à noter que la méthode Agent est la meilleure approche, et que l'ingénierie des prompts surperforme la méthode RAG.

Approche	LLM	Requête valide pour REST	Requête valide pour GraphQL
Prompt-Engineering	Llama3	0%	65%
	Mixtral 8x7B	0%	42%
	Codestral-22B	2%	69%
RAG	Llama3	0%	49%
	Mixtral 8x7B	0%	44%
	Codestral-22B	29%	56%
Agentic	Llama3	2%	73%
	Mixtral 8x7B	0%	72%
	Codestral-22B	**52**%	**77**%

TAB. 1 – *Validité de la syntaxe des requêtes générées en fonction des approches*

4.2 Détection des filtres pour les requêtes

Cette évaluation ne considère que les requêtes syntaxiquement valides. Comme observé dans la table 1, certaines méthodes génèrent peu ou pas de requêtes valides. Nous évaluons la capacité de ces méthodes à produire des requêtes similaires à celles des experts en comparant la liste des filtres. La table 2 permet d'observer que, même s'il est plus difficile de générer des requêtes REST syntaxiquement correctes que des requêtes GraphQL, les modèles obtiennent de meilleures performances sur les requêtes REST valides.

Pour évaluer les filtres, nous avons calculé les métriques suivantes : précision, rappel, exactitude et score F1. Pour cela, nous donnons les définitions suivantes. **TP** (True Positive) : les attributs sont présents dans les deux requêtes. **FP** (False Positive) : le modèle a généré des attributs supplémentaires. **FN** (False Negative) : le modèle a échoué à identifier des attributs. **TP-values** : les valeurs générées pour les attributs sont effectivement présentes dans la Land Matrix.

Approach	LLM	Precision		Recall		Accuracy		F1-score		Value-score	
		REST	GraphQL	REST	GraphQL	REST	GraphQL	REST	GraphQL	REST	GraphQL
Prompt-Engineering	Llama3	-	94%	-	55%	-	54%	-	70%	-	19%
	Mixtral 8x7B	-	92%	-	59%	-	40%	-	72%	-	12%
	Codestral-22B	100%	93%	100%	82%	100%	72%	100%	87%	2%	43%
RAG	Llama3	-	75%	-	18%	-	16%	-	29%	-	5%
	Mixtral 8x7B	-	40%	-	13%	-	10%	-	19%	-	2%
	Codestral-22B	67%	79%	61%	46%	46%	38%	64%	58%	40%	23%
Agentic	Llama3	100%	95%	36%	71%	36%	67%	53%	81%	2%	40%
	Mixtral 8x7B	-	81%	-	71%	-	60%	-	76%	-	33%
	Codestral-22B	88%	88%	74%	85%	67%	76%	80%	86%	59%	48%

TAB. 2 – *Précision des filtres dans les requêtes REST et GraphQL pour les requêtes syntaxiquement valides*

4.3 Exactitude des données retournées par l'API de LMI

Les requêtes générées peuvent légèrement différer de celles du corpus expert, mais elles peuvent, cependant, également récupérer l'ensemble de données attendu depuis l'API LMI. Ainsi, le tableau 3 présente la proximité Jaccard entre données issues de requêtes générées et celles issues des requêtes des experts.

L'analyse de ces résultats conduit à la même conclusion : bien que les modèles aient plus de difficultés avec la syntaxe REST, leurs requêtes valides performent mieux que celles pour GraphQL. La meilleure approche reste Codestral avec la configuration Agentic.

Approach	LLM	Valid Result for REST	Valid Result forGraphQL
Prompt-Engineering	Llama3	-	15%
	Mixtral 8x7B	-	2%
	Codestral-22B	2%	48%
RAG	Llama3	1%	5%
	Mixtral 8x7B	1%	3%
	Codestral-22B	56%	30%
Agentic	Llama3	2%	37%
	Mixtral 8x7B	2%	33%
	Codestral-22B	**66**%	**51**%

TAB. 3 – *Précision des données retournées par les requêtes générées en fonction des approches et des modèles*

5 Conclusion

La problématique abordée dans ce travail était de savoir si les nouvelles capacités de génération de code des LLMs pouvaient améliorer la tâche Text-to-SQL. Sur la base des données annotées par les administrateurs de la base de données Land Matrix, nous proposons une comparaison évolutive de trois modèles avec trois adaptations (Prompt, RAG et approche Agentic). La meilleure combinaison, à savoir Codestral en mode Agentic, atteint les deux tiers des résultats attendus par les utilisateurs de la base de données pour les requêtes REST et la moitié pour

les requêtes GraphQL. Ce champs de recherche étant très actif, nous avons veillé à ce que les expériences soient reproductibles, permettant ainsi d'ajouter d'autres concurrents à notre banc d'essai.

Remerciements

Cette étude a bénéficié du soutien financier de l'initiative Land Matrix ainsi que du soutien de la plateforme ISDM-MESO de l'Université de Montpellier, financée dans le cadre du CPER par l'État français, la Région Occitanie, la Métropole de Montpellier et l'Université de Montpellier.

Références

Anseeuw, W., M. Boche, T. Breu, M. Giger, J. Lay, P. Messerli, et K. Nolte (2012). *Transnational land deals for agriculture in the global south : analytical report based on the Land Matrix database*. CDE. CIRAD-ES-UMR ART-DEV (ZAF) ; CIRAD-ES-UMR ART-DEV (FRA).

Balaguer, A., V. Benara, R. L. d. F. Cunha, R. d. M. E. Filho, T. Hendry, D. Holstein, J. Marsman, N. Mecklenburg, S. Malvar, L. O. Nunes, R. Padilha, M. Sharp, B. Silva, S. Sharma, V. Aski, et R. Chandra (2024). RAG vs Fine-tuning : Pipelines, Tradeoffs, and a Case Study on Agriculture. arXiv :2401.08406 [cs].

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, et D. Amodei (2020). Language Models are Few-Shot Learners. arXiv :2005.14165 [cs].

Devlin, J., M.-W. Chang, K. Lee, et K. Toutanova (2019). BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv :1810.04805 [cs]*. arXiv : 1810.04805.

Hong, Z., Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, et X. Huang (2024). Next-generation database interfaces : A survey of LLM-based text-to-SQL.

Lewis, P., E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, et D. Kiela (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv :2005.11401 [cs].

Liu, S., J. Xu, W. Tjangnaka, S. J. Semnani, C. J. Yu, et M. S. Lam (2024). SUQL : Conversational Search over Structured and Unstructured Data with Large Language Models. arXiv :2311.09818 [cs].

Mohammadjafari, A., A. S. Maida, et R. Gottumukkala (2024). From natural language to SQL : Review of LLM-based text-to-SQL systems.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, et I. Polosukhin (2017). Attention Is All You Need. Number : arXiv :1706.03762 arXiv :1706.03762 [cs].

Zhao, W. X., K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, et J.-R. Wen (2023). A Survey of Large Language Models. arXiv :2303.18223 [cs].

Zheng, L., W.-L. Chiang, Y. Sheng, T. Li, S. Zhuang, Z. Wu, Y. Zhuang, Z. Li, Z. Lin, E. P. Xing, J. E. Gonzalez, I. Stoica, et H. Zhang (2023). LMSYS-Chat-1M : A Large-Scale Real-World LLM Conversation Dataset. arXiv :2309.11998 [cs].

Ziletti, A. et L. D'Ambrosi (2024). Retrieval augmented text-to-SQL generation for epidemiological question answering using electronic health records.

Summary

The Land Matrix initiative and its global observatory aim to provide reliable data on large-scale land acquisitions to inform debates and actions in sectors such as agriculture, extraction, or energy in low- and middle-income countries. Although these data are recognized in the academic world, they remain underutilized in public policy, mainly due to the complexity of access and exploitation, which requires technical expertise and a good understanding of the database schema.

The objective of this work is to simplify access to data from different database systems. The methods proposed in this article are evaluated using data from the Land Matrix. This work presents various comparisons of Large Language Models (LLMs) as well as combinations of LLM adaptations (Prompt Engineering, RAG, Agents) to query different database systems (GraphQL and REST queries). The experiments are reproducible, and a demonstration is available online: https://github.com/tetis-nlp/landmatrix-graphql-python.