IFT-6005 - Projet intégrateur - H25
Agent conversationnel pour l'interrogation de la base de données Open Food Facts
Description et planification du projet

réalisé par

Alain Boisvert

994 029 313

25 novembre 2024

IFT-6005 - Projet intégrateur - H25
Agent conversationnel pour l'interrogation de la base de données Open Food Facts

Alain Boisvert

23 janvier 2025

1 Description du problème

L'accès à des informations nutritionnelles fiables et la comparaison de produits alimentaires sont des enjeux importants pour les consommateurs.

« texte » ou utiliser le package csquotes

Il existe des bases de données ouvertes telles qu'Open Food Facts qui fournissent des informations sur les produits alimentaires, y compris les ingrédients, les valeurs nutritionnelles et les labels. Toutefois, l'accès à ces données est souvent difficile pour les utilisateurs non techniques. Les interfaces Web ou mobiles actuelles nécessitent souvent des recherches manuelles fastidieuses pour obtenir des informations précises sur les produits.

L'objectif de ce projet est de développer un agent conversationnel capable de répondre à des questions en langage naturel en interrogeant une base de données sur les produits alimentaires canadiens.

[TODO: Compléter cette section]

2 Revue de littérature

Résoudre ce problème nécessite un agent conversationel pour dialoguer avec l'utilisateur, une capacité d'interpréter les recherches de l'utilisateur et de générer des requêtes SQL sur ue base de données, ainsi qu'une capacité de synthèse des résultats pour les présenter à l'utilisateur.

L'approche RAG (Retriever-Reader-Generator) est une architecture populaire pour les agents conversationnels qui combine un modèle de recherche (Retriever), un modèle de lecture (Reader) et un modèle de génération (Generator). Conçu pour des données non structurées (textes, PDF, etc.), où la recherche par similarité sémantique (via embeddings) est efficace.

Les systèmes de RAG peuvent fonctionner avec des données structurées SQL, mais leur efficacité dépend de la manière dont les données sont préparées et intégrées au processus. Leur intégration dans RAG nécessite une transformation adaptée, par exemple :

- **Représentation textuelle** : Convertir les lignes de tables SQL en descriptions naturelles (ex : "Client : Jean, Commande : 123, Montant : 200€"). Cette approche peut ne pas être suffisante pour intégrer la structure relationnelle des données.
- **Requêtes SQL dynamiques** : Utiliser un LLM pour générer des requêtes SQL pertinentes, puis injecter les résultats dans le modèle génératif (approche hybride).

Ainsi, un LLM (Large Language Model) ayant la capacité d'utiliser un outil pour interroger une base de données semble être une solution appropriée.

Many modern LLMs (e.g., GPT-4, Claude) can use tools via post-training adaptations (e.g., plugins, function calling), but only a subset (like Qwen) were directly trained for this purpose during pre-training or instruction tuning.

Qwen-Chat has been optimized for tool usage and function calling capabilities. Users can develop agents, LangChain applications, and even augment Qwen with a Python Code Interpreter [1].

Les travaux suivants éclairent notre approche :

TODO : Cette section doit être bonifiée sinon complétée..

3 Approches proposées

Pour y parvenir, plusieurs approches sont possibles, chacune avec ses forces et ses limites.

3.1 Approche 1 : RAG classique

Le RAG classique repose sur l'analyse sémantique de données non structurées, comme des textes ou des documents. Un modèle de langue génère des réponses en s'appuyant sur des extraits pertinents trouvés dans ces données. Cette méthode fonctionne bien pour des questions ouvertes ou complexes, comme expliquer un concept à partir d'un manuel. Cependant, elle est peu adaptée aux bases de données structurées (ex : tables SQL), car elle ne gère pas efficacement les chiffres, les dates ou les relations entre tables.

Avantages Idéal pour des réponses contextuelles et naturelles.

Inconvénients : Perte de précision sur les données organisées en tables.

Approche 2 : Conversion des données SQL en texte)
Cette méthode transforme les lignes d'une base SQL en phrases simples (ex : « Client : Jean, Commande : 200€ ») pour les traiter comme du texte. Le modèle de langue peut alors utiliser ces phrases pour répondre aux questions. Cela permet d'exploiter des outils existants comme les chatbots, mais simplifie excessivement les relations entre les données (ex : jointures entre tables).

Avantages : Facile à mettre en place pour des schémas simples. **Inconvénients** : Risque de dégradation des résultats pour des bases volumineuses ou complexes.

Approche 3 : Génération dynamique de requêtes SQL)
Ici, un modèle de langue comme GPT-4 ou Mistral est entraîné à traduire une question utilisateur en requête SQL. Par exemple, la question « Quel est le chiffre d'affaires de juillet 2023 ? » devient 'SELECT SUM(montant) FROM ventes WHERE date BETWEEN '2023-07-01' AND '2023-07-31''. La requête est exécutée directement sur la base de données, garantissant des résultats précis.

Avantages : Exploite pleinement la structure des données (dates, catégories, etc.). **Inconvénients** : Dépend de la fiabilité du modèle pour générer des requêtes correctes.

Approche 4 : Utilisation d'Elasticsearch)
Elasticsearch est un outil de recherche qui combine filtres structurés (ex : « prix $>$ 500€ ») et recherche sémantique (ex : « produits populaires »). En indexant les données SQL dans Elasticsearch, l'agent peut répondre à des questions hybrides, comme « Quels clients mécontents ont dépensé plus de 1000€ en 2023 ? ». Cette approche offre flexibilité, mais nécessite une préparation minutieuse des données.

Avantages : Réponses riches, mélangeant chiffres et contexte sémantique. **Inconvénients** : Complexité d'intégration et maintenance accrue.

Dans le cadre du présent projet, je propose un développement incrémental basé sur ces approches, en commençant par une approche simple et évolutive vers des réponses plus nuancées.

1. See GitHub documentation

Étape 1 : Valider la génération de requêtes SQL
Pour commencer simplement, l'agent utilisera un modèle de langue (comme Mistral) pour convertir les questions en requêtes SQL. Une base légère, comme DuckDB, permettra d'exécuter ces requêtes sans infrastructure complexe. Par exemple, à la question « Quelles sont les ventes de décembre ? », le modèle générera 'SELECT * FROM ventes WHERE mois = '12''. Cette phase validera la capacité du système à comprendre les intentions de l'utilisateur et à produire des requêtes fiables.

Étape 2 : Intégrer Elasticsearch pour enrichir les réponses
Dans un second temps, les données SQL seront indexées dans Elasticsearch pour ajouter une couche sémantique. L'agent pourra alors répondre à des questions comme « Trouve les commandes récentes avec des retours négatifs ». Elasticsearch combinera automatiquement les filtres (ex : « date > 2023-01-01 ») et la recherche par similarité (ex : commentaires clients négatifs). Le modèle de langue synthétisera les résultats en une réponse claire, exploitant à la fois la structure des données et leur contexte.

tape 3 : Optimisation et tests utilisateurs
Enfin, le système sera testé avec des scénarios réels pour ajuster les prompts du modèle, corriger les requêtes incorrectes, et améliorer la fluidité des réponses. L'accent sera mis sur la gestion des erreurs (ex : requêtes SQL invalides) et la personnalisation des réponses.

Ce projet suit une logique de simplicité avant complexité : commencer par des requêtes SQL basiques pour valider le cœur du système, puis ajouter Elasticsearch pour des réponses plus nuancées. Cette approche minimise les risques tout en explorant des techniques variées, idéales pour un projet universitaire visant à concilier innovation et pragmatisme.

TODO : Cette section doit être bonifiée sinon complétée..

4 Choix du modèle LLM

Trois modèles seront évalués :

- **Mistral-7B** : À compléter
- **DeepSeek-R1-7B** :
- **Qwen-7B** :

TODO : Cette section doit être bonifiée sinon complétée..

Avantages comparatifs :

RAG	**Agents**
Accès précis aux données	Gestion du flux décisionnel
Maintien du contexte court	Coordination multi-étapes

5 Données utilisées

Le jeu de données Open Food Facts (version canadienne) contient :

- 94 782 produits alimentaires (mise à jour janvier 2024)
- 156 attributs par produit regroupés en 8 catégories :
  - Nutrition : 35 champs (dont 22% manquants)
  - Ingrédients : Composition textuelle (78% complète)
  - Labels : Certifications (Bio, Sans gluten...)

TODO : Cette section doit être bonifiée sinon complétée..

Table 1 – Exemple de données nutritionnelles

Produit	Calories (kcal)	Protéines (g)	Sucre (g)
Lait 2%	130	8	12
Pain complet	240	9	3
Yaourt nature	150	5	7

6 Évaluation du système

Métriques d'évaluation :

1. Exactitude des réponses (F1-score sur 500 questions de référence)
2. Temps de réponse moyen (objectif < 3s)
3. Taux de réussite multilingue (FR/EN)
4. Robustesse aux données manquantes

Méthodologie : Test A/B avec 50 utilisateurs et analyse des logs d'interaction.

TODO : Cette section doit être bonifiée sinon complétée..

7 Tâches à faire

Planification détaillée (270h total) :

— Intégration données (50h) : Nettoyage, validation, indexation
— Développement agent principal (70h) : Mécanismes de décision et gestion d'erreurs
— Implémentation RAG (80h) : Optimisation des embeddings et requêtes hybrides
— Tests performance (40h) : Benchmark sur 4 configurations matérielles
— Documentation (30h) : Guide technique et manuel utilisateur illustré

TODO : Cette section doit être bonifiée sinon complétée..

LaTeX

Comme démontré par [6], les méthodes d'évaluation agentiques...

Références

[1] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces : A survey of llm-based text-to-sql. *arXiv preprint arXiv :2406.08426*, 2024.

[2] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface ? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.

[3] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench : Evaluating llms as agents. *arXiv preprint arXiv :2308.03688*, 2023.

[4] Ali Mohammadjafari, Anthony S Maida, and Raju Gottumukkala. From natural language to sql : Review of llm-based text-to-sql systems. *arXiv preprint arXiv :2410.01066*, 2024.

[5] Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. *arXiv preprint arXiv :2410.07869*, 2024.

[6] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents : A survey. *arXiv preprint arXiv :2309.07864*, 2023.

[7] Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. Synthesizing text-to-sql data from weak and strong llms. *arXiv preprint arXiv :2408.03256*, 2024.

[8] Fan Zhang, Shulin Tian, Ziqi Huang, Yu Qiao, and Ziwei Liu. Evaluation agent : Efficient and promptable evaluation framework for visual generative models. *arXiv preprint arXiv :2412.09645*, 2024.