Conversational Geographic Question Answering for Route Optimization: An LLM and Continuous Retrieval-Augmented Generation Approach

JOSE TUPAYACHI* and XUEPING LI*, University of Tennessee, USA

We investigate the capabilities of Large Language Models (LLMs) to interact with application programming interfaces based on documentation and logical instructions, specifically in the context of Geographic Question Answering for route optimization. A Continuous Retrieval-Augmented Generation method combined with LLMs, where customized node-based storage is implemented, followed by vector search retrieval. A comparative analysis of Gemma 2 and LLaMA 3.1 models demonstrates the method's potential and flexibility in handling complex queries. The paper's extension will evaluate the framework using the METEOR [1] and BERTScore [2] metrics, providing insights into their specialized knowledge and performance.

CCS Concepts: • **Information retrieval**; • **Generative models**; • **Geographic information systems**; • **Language models**;

ACM Reference Format:
Jose Tupayachi and Xueping Li. 2024. Conversational Geographic Question Answering for Route Optimization: An LLM and Continuous Retrieval-Augmented Generation Approach. In *17th ACM SIGSPATIAL International Workshop on Computational Transportation Science GenAI and Smart Mobility Session (IWCTS'24), October 29-November 1 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, Article 111, 6 pages. https://doi.org/10.1145/3681772.3698217

1 INTRODUCTION

Recent advancements in Natural Language Processing (NLP) and Large Language Models, known as transformers, have been adapted across various fields [Chang et al. 2023]. Knowledge Graphs (KGs) aid in converting complex data into organized network entities, thereby enabling users to query and extract information effectively. The integration of KGs can improve search engines by analyzing both semantic and structured data. Relationships and semantic connections excel at representing complex domains linking data through ontologies [Ji et al. 2021]. However, as the scale and diversity of information grows, there is an increasing need for more flexible, efficient, and contextually aware retrieval methods. Vectorized databases represent a significant evolution in information storage and retrieval when paired with solutions like vector search [Kersten et al. 2018]. Encoding data into high-dimensional vectors, enables advanced search capabilities beyond keyword matching, allowing for similarity-based retrieval using embedding [Reimers and Gurevych 2019]. This shift handles unstructured data more effectively, suitable for topic-specialized chatbots that can access and process vast amounts of information retrieving facts based on similarity.

*Both authors contributed equally to this research.
[1]Metric for Evaluation of Translation with Explicit ORdering
[2]Bidirectional Encoder Representations from Transformers

IWCTS'24, October 29-November 1 2024, Atlanta, GA, USA

ACM ISBN 979-8-4007-1151-0/24/10…$15.00
https://doi.org/10.1145/3681772.3698217

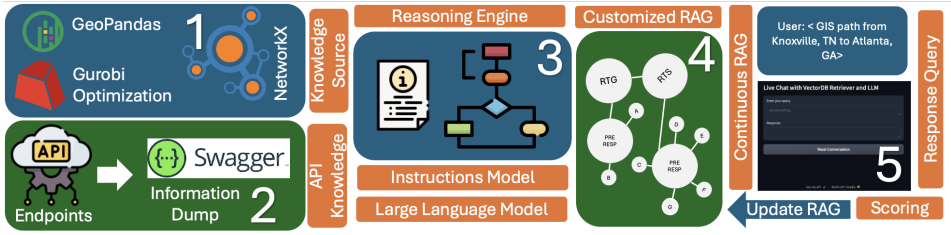

Fig. 1. Overview of CRAG, architecture and usage

Vectorized storage, search, and retrieval are built on the foundations of knowledge graphs but extend their capabilities in dynamism and scalability. Question Answering (QA) is an emerging field with the support of LLMs whose goal is to generate automated answers for questions asked in natural language [Zhuang et al. 2024]. Yet, spatial information poses higher challenges such as the calculation of spatial proximity between two locations [Mai, Janowicz, R. Zhu, et al. 2021]. These difficulties primarily arise because polygon-based geographic operations are computationally expensive. Geographic Question Answering (GQA) focuses on the development of systems that can respond to questions containing geographical information with high accuracy [Mai, Janowicz, Cai, et al. 2020]. Utilizing REST API [3] as a point of knowledge distribution grants access to vast pools of specialized data. However, effectively leveraging this information often requires adherence to rigid rules dictating how it should be used. Time-consuming and complex when addressing diverse queries. The interactive use of LLMs introduces flexibility, thanks to their capabilities. Despite their potential, most solutions still rely on pre-trained extensively labeled data, which can restrict applicability. Retrieval-augmented generation addresses the limitations of pre-trained models, which often demand substantial hardware resources. By leveraging external knowledge sources, RAG reduces these dependencies, enabling more flexible knowledge. Our interactive approach employs human-in-the-loop techniques and a learned reasoning to understand and provide accurate responses, with specialized information sourced from API endpoints. The Continuous RAG (CRAG) uses a semi-structured pipeline, continuously updating its knowledge base. This iterative process ensures that RAG maintains up-to-date and tuned responses, as explained under Algorithm 1.

2 METHOD

Specialized knowledge demands extensive training in specific domains, along with advanced reasoning capabilities [Gao et al. 2023]. Techniques like RAG can provide domain-specific responses. The work by [Ye et al. 2024], has demonstrated innovative approaches to question-answering GIS data. ConvRAG integrates question refinement, RAG, and a self-check mechanism to grant improved question comprehension and information retrieval in conversational environments. We leverage external databases through a REST API and incorporate APIs and documentation for query generation. [Gamage et al. 2024] employs a multi-format data pipeline. Unstructured text data is used to build reasoning upon a set of endpoints API of documentation through swagger [SmartBear 2024]. Similar research focuses on fine-tuning or domain-specific initial knowledge sources by leaking a priori the domain of the tested LLM subject [Tianjun Zhang et al. 2024]. We implement a continuous RAG approach which iteratively refines the information by including it in the searchable data and later responses.

Starting on the backend API and JSON documentation, this dictates the query syntax and provides a structured knowledge source. Each endpoint documentation includes its functions and required

[3]Representational State Transfer Application Programming Interface

parameters. This documentation serves as a foundational resource for knowledge sources in the retrieval enabling the generation of specialized responses for each query. In Figure 1, the initial source of information consists of a set of proceeded geographical data optimized for shortest path and route calculations using GeoPandas for GIS parsing and NetworkX for shortest path determination and neighboring location. This data is stored in PostgreSQL and endpoints are created in Django. Documentation is generated through Swagger and then parsed into nodes. Node information and attributes are stored using the pgvector add-on. The approach utilizes a conversational pipeline (interactive) to determine the correct endpoint, request all parameters, perform the search based on prior input, and retrieve cURL results. Historical chats enhance the information iteratively until the final retrieval is completed. A Python-based driver or executor runs the curl commands and interfaces with LLM. This output is then inserted the Continuous RAG as in Algorithm 1 that continuously updates the vectorized database for the subsequent queries generating specialized responses. The application offers two types of endpoints: unique endpoints, which list all available options, and raw data endpoints, which provide unfiltered, datasets. Five endpoints LCA (Life Cycle Analysis), RTS (Neighboring Cities), RTG (Routes, GIS paths, and distances), CRD (Points of Interest), and DMD (Freight Demand Requirements).

2.1 Storage & Retrieval:

The `all-MiniLM-L6-v2` model uses 6 layers and small hidden sizes. Carries a self-attention mechanism in computing a weighted sum of representations of all words in a sentence. Input tokens are converted into dense vectors through multiple layers of self-attention and feed-forward networks. Each layer applies a linear transformation and non-linear activation functions. Contextualized embeddings for each token generate a single embedding for the entire sentence. Here, $\mathbf{e}_i$ represents the embedding of the i-th token, and N is the total number of tokens in the sentence. The model includes customized metadata that consists of the type of source, node identifier, length, tags, positional order, and relationship. The semantic similarity between text elements is determined by measuring distances or similarities between their vector representations. Information is indexed based on their embeddings. When a query is made, its embedding is compared to the embeddings of documents in the index to find the most similar documents. This allows for efficient and scalable retrieval based on semantic similarity. We employ pgvector [Kane Andrew 2024] and it is set to the nearest neighbor search algorithm providing perfect recall. Let $\mathbf{E} \in \mathrm{R}^d$ represent the given query embedding. Let $\mathbf{e_i} \in \mathrm{R}^d$ represent the stored embedding for the i-th record in the table. The cosine distance D($\mathbf{E}$, $\mathbf{e_i}$) between $\mathbf{E}$ and $\mathbf{e_i}$ is defined as: $D(\mathbf{E}, \mathbf{e_i}) = 1 - \frac{\mathbf{E} \cdot \mathbf{e_i}}{\|\mathbf{E}\| \|\mathbf{e_i}\|}$ where $\mathbf{E} \cdot \mathbf{e_i}$ is the dot product between the query embedding E and the stored embedding $\mathbf{e_i}$, and $\|\mathbf{E}\|$ and $\|\mathbf{e_i}\|$ are the magnitudes of the embedding. The query retrieves the records r_i (with attributes such as id, $node_id$, text, metadata, and the embedding) from the database table, calculating the cosine distance D(E, $\mathbf{e_i}$) for each record. The records are then sorted by their cosine distance D(E, $\mathbf{e_i}$) in ascending order, with the top `limit` records being selected. If `metadata_filters` are provided, an additional filter condition is applied to the query to restrict the results based on the metadata. The query retrieves up to `limit` records r_i such that the cosine distance D(E, $\mathbf{e_i}$) is minimized, and the records satisfy any given `metadata_filters`.

2.2 Continuous RAG (CRAG) Architecture

The RAG procedure separates stored utilizing an optimized vector search function, after which the generative and attention-based mechanisms of the LLM permit the creation of specific query sentences that respond to user needs. Process orchestration is required between the LLM, backend, and user, a wrapper component is necessary. This entity receives user inquiries, sends prompts

to collect specified parameters, and permits specifying the required model. The generated output is stored in the relational database and subsequently linked with metadata to enable the models to understand the underlying logic for each API as observed in Algorithm 1. This addresses the current limitations of the *context_window*, which is notably constrained even in large models like LLaMA-3.1 405b [Dubey et al. 2024]. We offer a near-unlimited context window by first refining the context with a feedback mechanism and then utilizing it in the model's operations.

3 COMPUTATION & EXPERIMENTATION

We evaluate the performance of leading models and compare them to those results obtained by manual execution. This provides a comprehensive assessment of their efficacy in retrieving and answering geographical-related information. We evaluate a set of local large language models. Specifically, we use LLaMA 3.1 8b and Gemma 2 9B IT. We utilize a set of pre-calculated information to determine whether each model properly queried and outputted the expected values after the interactive approach. Testing is performed by the use of Selenium web driver [SeleniumHQ 2024] following automated testing.

Model performance is aimed to be tracked using quantitative metrics such as BERTscore [Tianyi Zhang et al. 2019] and METEOR [Banerjee and Lavie 2005], alongside error analysis. By integrating expert and user feedback and leveraging automated systems for continuous learning the CRAG training process and feedback utilized the Gemma model, known for its structured output.

Figure 1 details the use of single-parameter queries, as opposed to two-parameter queries, which appears to complicate the instructional query and retrieval process, making it more challenging to follow. For 1 parameter scenario, the hub or city name is considered. Notably, the application of the CRAG method demonstrates a significant improvement in the Gemma-based model compared to the Llama model in CRD. Nevertheless, a llama-based model presented a slight decrement in regards to the BERTscore metric during LCA testing. The outcome in the METEOR model exhibits better performance for both CRD and LCA independently in the majority of the models. Experiments utilized a temperature setting of 0.2, a maximum of 256 new tokens, a context window of 1024, and embeddings with a dimensionality of 384. Whereas longer models utilize a maximum of 1024 tokens for generating new content, they are also configured with a context window of 3084 tokens. CRAG training used 20 samples and testing was performed on 30 per endpoint.

4 CONCLUSIONS & FUTURE DIRECTIONS

The proposed approach, when combined with CRAG techniques, aims at ensuring accurate responses. Our workflow begins with a base of information, which is then enhanced by leveraging the output of an LLM, vectorized, and stored. This integration of LLM and RAG creates a powerful platform capable of executing API requests across various instances. Future directions will focus on utilizing more comprehensive node metadata and provide an extensive comparison considering a higher complexity request to evaluate the impact on the methodology. These while pairing a direct shell access for query validation.

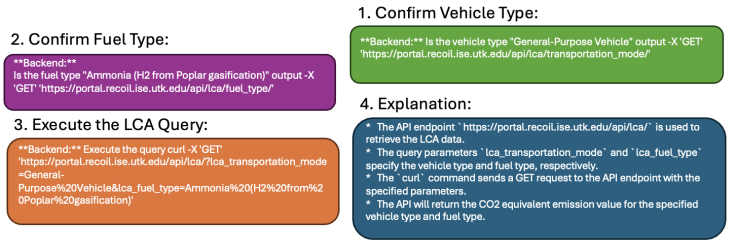

Fig. 2. Expected output including cURL instruction.

REFERENCES

Satanjeev Banerjee and Alon Lavie. 2005. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments." In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 65–72.

Yupeng Chang et al.. 2023. "A survey on evaluation of large language models." *ACM Transactions on Intelligent Systems and Technology*.

Abhimanyu Dubey et al.. 2024. "The llama 3 herd of models." *arXiv preprint arXiv:2407.21783*.

Gihan Gamage, Nishan Mills, Daswin De Silva, Milos Manic, Harsha Moraliyage, Andrew Jennings, and Damminda Alahakoon. 2024. "Multi-Agent RAG Chatbot Architecture for Decision Support in Net-Zero Emission Energy Systems." In: *2024 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 1–6.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. "Retrieval-augmented generation for large language models: A survey." *arXiv preprint arXiv:2312.10997*.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. 2021. "A survey on knowledge graphs: Representation, acquisition, and applications." *IEEE transactions on neural networks and learning systems*, 33, 2, 494–514.

Linnakangas Heikki Kane Andrew. 2024. *pgvector: Open-source extension for vector similarity search in PostgreSQL*. https://github.com/pgvector/pgvector. GitHub repository. (2024). https://github.com/pgvector/pgvector.

Timo Kersten, Viktor Leis, Alfons Kemper, Thomas Neumann, Andrew Pavlo, and Peter Boncz. 2018. "Everything you always wanted to know about compiled and vectorized queries but were afraid to ask." *Proceedings of the VLDB Endowment*, 11, 13, 2209–2222.

Gengchen Mai, Krzysztof Janowicz, Ling Cai, Rui Zhu, Blake Regalia, Bo Yan, Meilin Shi, and Ni Lao. 2020. "SE-KGE: A location-aware knowledge graph embedding model for geographic question answering and spatial semantic lifting." *Transactions in GIS*, 24, 3, 623–655.

Gengchen Mai, Krzysztof Janowicz, Rui Zhu, Ling Cai, and Ni Lao. 2021. "Geographic question answering: challenges, uniqueness, classification, and future directions." *AGILE: GIScience series*, 2, 8.

Nils Reimers and Iryna Gurevych. Nov. 2019. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, (Nov. 2019). https://arxiv.org/abs/1908.10084.

SeleniumHQ. 2024. *Selenium WebDriver*. Accessed: [09-06-2024]. (2024). https://www.selenium.dev/documentation/webdriver/.

SmartBear. 2024. *Swagger: API Documentation Design Tools*. https://swagger.io. Version 3.0.0. (2024).

Linhao Ye, Zhikai Lei, Jianghao Yin, Qin Chen, Jie Zhou, and Liang He. 2024. "Boosting Conversational Question Answering with Fine-Grained Retrieval-Augmentation and Self-Check." In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2301–2305.

Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. 2024. "Raft: Adapting language model to domain specific rag." *arXiv preprint arXiv:2403.10131*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. "Bertscore: Evaluating text generation with bert." *arXiv preprint arXiv:1904.09675*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024. "Toolqa: A dataset for LLM question answering with external tools." *Advances in Neural Information Processing Systems*, 36.

Received 10 September 2024; revised 25 September 2024; accepted 6 October 2024

Algorithm 1 Conversational Pipeline: Standardized Input-Output Interactive W/O CRAG Training

Input: User query Q (e.g., `Calculate the distance between two cities.`)
Step 1: User Request; $Q = f(\text{Input})$; $Q \in \mathcal{D}$ where $\mathcal{D}$ represents the domain of possible user queries from documentation.
Step 2: Query Determination; Determine API endpoint A_Q for query Q based on rule set $\mathcal{R}$; $A_Q = g(Q, \mathcal{R})$; (e.g., if Q relates to geographic data, then $A_Q \in$ <api/rtg/>)
Step 3: Machine Action (Search for API); Extract parameters from user input $P = \{p_1, p_2, \ldots, p_n\}$ from Q for A_Q
if $P = \emptyset$ **then**
    Ask user for input parameters $P = \{p_1, p_2, \ldots, p_n\}$
end if
    Execute API call via curl command; $R_{\text{raw}} = \texttt{curl}(A_Q, P)$
Step 4: Backend Processing; Insert queried data R_{raw} into the RAG system; $R_{\text{refined}} = h(R_{\text{raw}}, \mathcal{M})$, where $\mathcal{M}$ is the RAG model.
Step 5: Machine Action (Response Generation); Return the refined response R based on R_{refined}: $R = \text{GenerateResponse}(R_{\text{refined}})$
Step 6: Response Validation
while R does not meet expected criteria (e.g., incomplete, incorrect) **do**
    Go back to Step 3 and repeat the process;
end while
Output: Final response R (e.g., `The distance from city A to city B is X miles.`)

Table 1. Performance Metrics for CRD and LCA