GraphAide:Advanced Graph-Assisted Query and Reasoning System

Sumit Purohit, George Chin
Pacific Northwest National Laboratory
Richland, Washington 99352
{sumit.purohit,george.chin}@pnnl.gov

I. Abstract

Pattern matching and querying are fundamental applications of modern data analytics. The development of these applications involves addressing multiple research challenges such as data curation and extraction, named entity recognition, data modeling, and query interface development. Additionally, the explainability of such capabilities is crucial for their wider adoption. The advent of Large Language Models (LLMs) has accelerated the development lifecycle of new capabilities. However, there is a need for domain-specific tools optimized for user activities. The development of digital assistants has gained traction over the years, and LLMs present an opportunity to build such assistants using domain-specific knowledge and assumptions. We present an advanced query and reasoning system, *GraphAide*, that leverages knowledge graphs and LLMs to quickly develop domain-specific digital assistants. GraphAide combines design patterns from retrieval augmented generation (RAG) and the semantic web to develop an agentic LLM application.

II. Introduction

Large Language Models (LLMs) [2] represent the forefront of artificial intelligence and machine learning research and development. They have been adopted at an unprecedented rate across diverse fields such as scientific research, engineering, economics, and social sciences. By enabling non-computer science experts to utilize pre-trained, out-of-the-box models, LLMs have democratized application development, fostering innovation in domain-specific tasks. The development of LLM-based applications remains a vibrant area of research, with demonstrated superior performance in tasks such as summarization, correlation, and inference across multiple input sources [8].

One of the major challenges in developing accurate and consistent capabilities based on LLMs is *hallucination* [7], which is defined as the fabrication of non-existent facts in response to user queries. The memorization of training data by LLMs and the subsequent use of corpus-based heuristics are significant factors contributing to hallucination [5].

Retrieval-augmented generation (RAG) [4] is an established design pattern to address hallucination by providing additional context to the LLM in generating responses to user queries. The provided context is domain-dependent and serves as *grounding* for the model's generative capabilities. A baseline RAG-based system leverages a vector database to store *embeddings* of a domain-specific knowledge corpus and utilizes semantic similarity algorithms to extract the *context* relevant to the user query. This context is combined with the user query and sent to the LLM, acting as a guardrail for its generation process. The RAG-based approach has been shown to reduce hallucination by constraining the LLM's generative process to the localized region of the combined prompt.

However, a vector search-based approach is limited by the semantic similarity between the query and the corpus. It does not leverage the structural context, such as the relationships between documents in the corpus and their metadata, or the reasoning associated with it. Knowledge Graphs (KGs) offer a robust solution to these limitations by efficiently storing domain-specific information and creating relationships between information sources scattered across documents in the corpus. KGs can also enhance various components of an LLM-based application, including retrieval, summarization, and inference.

Existing KG-based LLM capabilities are application-focused [3, 6, 1] and do not leverage state-of-the-art open-source application development frameworks. There is a significant opportunity to utilize KGs more effectively in various phases of LLM-based application development and end-use.

We present GraphAide, a cutting-edge capability based on Large Language Models (LLMs) that provides insights into domain-specific data and allows users to ask natural language questions. GraphAide employs a modular, extensible, and user-friendly retrieval-augmented generation (RAG) approach, incorporating both vector and graph databases.

The novelty of GraphAide lies in its end-to-end platform, which curates multi-source data and utilizes knowledge graphs to address many challenges faced by baseline LLM-only applications. The platform also provides capabilities to persist data efficiently, enabling interactions via a natural language question-answering interface. Additionally, GraphAide combines a powerful graph matching approach with the semantic similarity provided by a vector database to increase the accuracy and explainability of responses generated by LLMs. GraphAide is developed as an extensible agentic application that reuses data ingestion, entity disambiguation, storage, query, and reasoning components at various phases of the application lifecycle.

GraphAide provides the following key capabilities:

1) Constructs a multi-value, multi-source, temporal knowledge graph from common data and file formats such as .txt, .html, .pdf, etc.
2) Generates low-dimension representations of the input data, also known as embeddings, and configures vector databases to store these embeddings. GraphAide uses these embeddings to support semantic similarity features.
3) Supports domain-specific and user-provided ontologies to guide knowledge graph generation. GraphAide employs a template-based approach to dynamically configure input ontologies and generate multiple KG versions for a given source dataset.
4) Enables natural language user queries and executes them as subgraph-matching operations.
5) Supports temporal retrieval using time-series databases such as TimeScaleDB.

III. Architecture

Agent-based LLM applications leverage the reasoning capabilities of an LLM to identify the appropriate sequence of actions to respond to user queries. In contrast to hardcoded instructions (i.e., *chains*), an *agentic application* is capable of understanding the LLM's response and constructing the next query for the LLM. GraphAide is a hybrid application that leverages both chains and agents to support various user queries. GraphAide defines *phases* corresponding to a set of features required to perform a user task. As shown in Figure 1, GraphAide has a *curation* phase that combines information from multiple sources to produce a KG, serving as a knowledge repository in addition to the vector database. Similarly, GraphAide has an *exploration* phase that provides a user interface to query the knowledge sources and produce responses in a user-provided format, combined with explanations and reasoning.

A. Data Ingestion

GraphAide supports commonly used data formats such as TEXT, CSV, PDF, etc. A collection of source data (i.e., files) is ingested into GraphAide, following a process of text splitting and chunking to support efficient search over vector data and to handle data larger than the maximum context window size (in number of tokens) of an LLM. GraphAide uses the $RecursiveCharacterTextSplitter$, which recursively examines characters in the input data to dynamically identify the best splitting strategy. GraphAide uses optimal $chunk_size$ and $chunk_overlap$, as suggested in published literature, and also allows users to configure the text splitter for a specific application domain.

B. Data Storage

After splitting and chunking, the source data is converted into a collection of $documents$, where each $document$ serves as an atomic knowledge source used to answer user queries. For a RAG-based LLM application, *embedding*-based semantic similarity is a key feature. Embeddings are numerical representations of information, such that semantically similar content is placed closer together in the embedding space. GraphAide uses a vector database to store embeddings of domain-specific reference datasets or general-purpose knowledge bases, such as WikiData. GraphAide provides a simple API to ingest datasets into the vector database and persist them locally for future use.

The novelty of GraphAide lies in its use of a graph database to store additional structural information. This graph database is employed in subsequent phases to enhance the performance of retrieval operations.

C. Knowledge Graph Generation

A key feature of GraphAide is the construction of a knowledge graph from curated sources. GraphAide employs a prompt-based, few-shot approach to instruct an LLM to create a knowledge graph. The GraphAide prompt takes an ontology or a list of node and edge types as input to restrict the LLM's response, ensuring it generates an attributed subgraph for a given input query. An example input query for the curation phase could be a paragraph from a news article.

D. Subgraph Query Generation

GraphAide combines expressive subgraph matching with vector-based semantic similarity to enhance retrieval accuracy. As shown in Figure 1, GraphAide leverages LLMs to convert user-provided natural language queries into attributed subgraph matching queries.

The GraphAide API supports commonly used graph databases, such as Neo4J, and can auto-generate Cypher queries. Additionally, the GraphAide API provides prompts to generate queries for domain-specific graph databases based on custom schemas.

GraphAide also performs query expansion to generate multiple versions of user-provided queries. The resulting query library is converted into subgraph queries and executed against the graph database.

E. Explainable Result Generation

GraphAide uses an advanced RAG-based approach to combine graph-based matching results with vector-based semantic similarity to prepare local context for the LLM. GraphAide executes multiple parallel requests to the LLM for the expanded query set. It makes a secondary call to the LLM to aggregate all the responses and generate a single coherent response for the given user query.

Research Challenges and Opportunities:

1) Dynamic Query Generation: Supporting dynamic query generation with custom edge type (relationship) schemas.
2) Cypher Query Generation Limitations: Current Cypher query generation focuses only on the most specific concepts and does not create multiple clauses, limiting its utility.
3) SummaryKGQuestion: Investigating the creation of Cypher queries from user questions that run against the knowledge graph (KG).

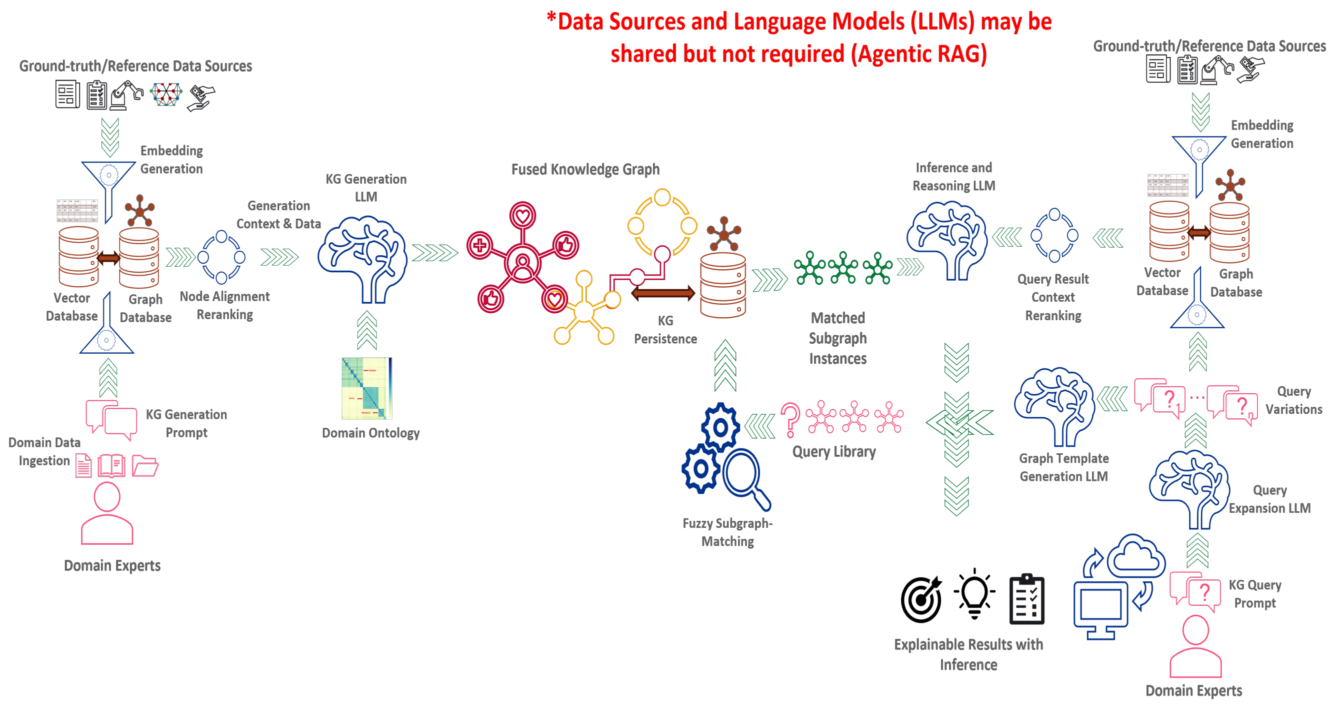

Fig. 1. GraphAide Architecture, representing different phases and agentic workflow

4) Framework Debugging: The framework is difficult to debug due to overloaded methods, variables, and extensive abstraction.

IV. Conclusion

LLMs offer an exciting opportunity to develop applications that address explainability, scalability, and usability concerns in using AI/ML. GraphAide leverages knowledge graph and subgraph matching capabilities to improve the accuracy of RAG-based applications. We present the reference architecture and its key components.

Acknowledgement

The research described in this paper is supported by the Resilience Through Data Driven, Intelligently Designed Control (RD2C) Initiative at Pacific Northwest National Laboratory (PNNL).

References

[1] Taiyu Ban, Lyvzhou Chen, Xiangyu Wang, and Huanhuan Chen. From query tools to causal architects: Harnessing large language models for advanced causal discovery from data. *arXiv preprint arXiv:2306.16902*, 2023.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[4] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[5] Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. Sources of hallucination by large language models on inference tasks. *arXiv preprint arXiv:2305.14552*, 2023.

[6] Milena Trajanoska, Riste Stojanov, and Dimitar Trajanov. Enhancing knowledge graph construction using large language models. *arXiv preprint arXiv:2305.04676*, 2023.

[7] Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Mu-Nan Ning, and Li Yuan. Llm lies: Hallucinations are not bugs, but features as adversarial examples. *arXiv preprint arXiv:2310.01469*, 2023.

[8] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.