GraphLingo: Domain Knowledge Exploration by Synchronizing Knowledge Graphs and Large Language Models

Duy Le Kris Zhao Mengying Wang Yinghui Wu
Case Western Reserve University
Cleveland, USA
{dhl64, kxz167, mxw767, yxw1650}@case.edu

***Abstract*—Knowledge graphs (KGs) are routinely curated to provide factual data for various domain-specific analyses. Nevertheless, it remains nontrivial to explore domain knowledge with standard query languages. We demonstrate GraphLingo, a natural language (NL)-based knowledge exploration system designed for exploring domain-specific knowledge graphs. It differs from conventional knowledge graph search tools in that it enables an interactive exploratory NL query over domain-specific knowledge graphs. GraphLingo seamlessly integrates graph query processing and large language models with a graph pattern-based prompt generation approach to guide users in exploring relevant factual knowledge. It streamlines NL-based question & answer, graph query optimization & refining, and automatic prompt generation. A unique feature of GraphLingo is its capability to enable users to explore by seamlessly switching between a more 'open' approach and a more relevant yet 'conservative' one, facilitated by diversified query suggestions. We show cases of GraphLingo in curriculum suggestion, and materials scientific data search.**

I. Introduction

Various domain-specific knowledge graphs (KGs) [1] [2] have been curated to host factual knowledge about specific topics rather than generic Web or common knowledge. Notable examples include material science, healthcare and disease, education, cybersecurity, biology, and chemistry. While knowledge curation has been extensively studied, searching and annotating domain data remains nontrivial. Domain experts are still expected to write complex declarative queries (such as SPARQL), or data scripts to parse their requests, in order to access the KGs. There is a gap between the need of accessing KGs with (domain) languages and optimized performance of query processing within state-of-the-art KG data systems.

The emergence of large language model (LLM), such as GPT [3], provides promising capabilities in generating natural language solutions in response to users' prompts. Although desirable, LLMs often fall short at verifying the truth of the generated results and may produce false statement that do not reflect commonsense or scientific facts (known as "Hallucination"). Furthermore, they have limited capacity to reason without proper contextualized domain knowledge.

In response, the possibility of bridging KGs and LLMs has attracted increasing interest. For example, advanced LLMs like GPT-4 [3] and PaLM [4] is allowed to browse Web knowledge and learn from a broader context by recent effort studying the coupling of knowledge graphs and LLMs [5]. On one hand, LLMs can be enhanced with KGs to provide answers with more contextualized facts. On the other hand, fundamental tasks such as KG curation, embedding, and search can also benefit by adopting LLMs. *Can we have a system that marries the merits of both directions, to enable domain knowledge exploration with natural language style Q&A?*

GraphLingo. We demonstrate GraphLingo, a system that *synchronizes* domain-specific KGs and LLMs to guide users in exploring domain knowledge with natural language. It differs from prior systems with the following unique features.

Automatic Prompt Generation. GraphLingo uses an automatic prompt generation algorithm to bridge NL responses of LLMs and graph search. Inspired by Query-By-Example (QBE) [6], it transforms NL questions into triple templates (graph pattern queries), and jointly exploit graph topological properties and textual representations as "examples" to generate in-context prompts and request NL responses from LLMs.

Preference-aware Exploration. GraphLingo allows users to pose ad-hocly a tunable preference, to explore factual knowledge (a) in a more "explorative" manner with open-ended questions and results from external Web knowledge in LLM, or (b) be more "conservative", favoring more in-context questions and answers in KG search. This is enabled by a diversified prompting strategy which leads to the generation of more in-context or open-end queries and answers. The desired feature in turn lets users explicitly control their knowledge preference.

Visual Exploration with Graph Views. GraphLingo supports a dual-design of interface that illustrate (1) NL-based conversational interface, and simultaneously, (2) a fraction of facts that are responsible for the generated results. This allows a visual explanation of the NL-based question & answer process.

Scalability. GraphLingo adopts a parallel computation friendly design and allow effective multi-query optimization and parallelism, to support large-scale diversified exploration efficiently.

A proof-of-concept system of GraphLingo is available [7]. We next provide an overview of its workflow with major modules (Section II) and the architecture (Section III).

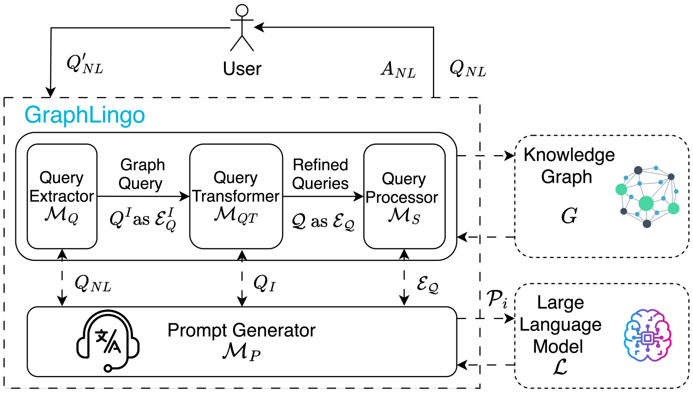

Q'_{NL}
User
A_{NL}
Q_{NL}
GraphLingo
Query Extractor $\mathcal{M}_Q$
Graph Query
Q^I as $\mathcal{E}^I_Q$
Query Transformer $\mathcal{M}_{QT}$
Refined Queries
$\mathcal{Q}$ as $\mathcal{E}_Q$
Query Processor $\mathcal{M}_S$
Knowledge Graph G
Q_{NL}
Q_I
$\mathcal{E}_Q$
$\mathcal{P}_i$
Prompt Generator $\mathcal{M}_P$
Large Language Model $\mathcal{L}$

Fig. 1: GraphLingo Workflow

II. Framework Overview

We start with several notations used by GraphLingo. (1) A (knowledge) *graph* G is a set of triple statements E in the form of $<s, r, o>$, where s and o are two (attributed) nodes that refer to two real world entities, and r is a relation between s and o. (2) A *graph pattern query* is a graph Q as a set of *triple template*. A triple template e_Q parameterizes a triple $<s, r, o>$ by assigning one or all of s, r and o a variable (denoted by ?, *e.g.,* $?s$) to indicate an output set, which refers to the "matches" of e_Q in terms of graph pattern matching such as subgraph isomorphism. (3) A *language model* LLM $\mathcal{L}$ is a function that takes as input a *prompt* and generates a natural language answer A_{NL}. (4) A *prompt* P is a NL statement that follows a general template of *context description*, *a set of examples*, and *a task description*. GraphLingo generates prompt templates by instantiating a set of prompts that specify NL statements and triple expressions.

Example 1: Consider the task that requests to transform an NL statement to a graph pattern query. A fraction of prompt $\mathcal{P}_1$ generated by GraphLingo is shown below[1]:

> Example: *'What are the topics of CSDS 310?'* has an equivalent graph query: [['?:Topic','of', 'CSDS 310']].
> Task: What is the graph query equivalent to '*What professor is teaching Data Mining?*'

Setting LLM as ChatGPT, we get correct answer as "[['?:Professor','teaches', 'Data Mining']]".

A. GraphLingo *Workflow*

GraphLingo works with a domain-specific knowledge graph G and a pre-trained LLM model $\mathcal{L}$, and four major functional modules that enables multi-session domain knowledge exploration. Each session starts with an input NL question Q_{NL} from a queryer, and performs three steps to yield a set of NL answers A_{NL}, along with a set of *suggested NL questions* Q'_{NL}, to promote the next session of exploration.

Query Extraction. Upon receiving a NL query Q_{NL}, GraphLingo uses a QueryExtractor module (denoted as $\mathcal{M}_Q$) to translate Q_{NL} into an initial graph pattern query Q^I with a set of triple templates $\mathcal{E}^I_Q$. This step incurs the first round of communication between module $\mathcal{M}_Q$ and the LLM model $\mathcal{L}$ (*e.g.,* ChatGPT), which (1) consults a bridging *prompt generator* modular $\mathcal{M}_P$ to produce a set of prompts $\mathcal{P}_1$, and (2) feed $\mathcal{P}_1$ to $\mathcal{L}$ to obtain Q^I in the form of $\mathcal{E}^I_Q$.

$$\boxed{\begin{aligned} \mathcal{P}_1 &\leftarrow \mathcal{M}_Q(\mathcal{M}_P, Q_{NL}) \\ \mathcal{E}^I_Q &\leftarrow \mathcal{L}(\mathcal{P}_1) \end{aligned}}$$

Query Transformation. Upon receiving graph query Q^I, GraphLingo next invokes a QueryTransformer module, denoted as $\mathcal{M}_{QT}$, to transform Q^I (as $\mathcal{E}^I_Q$) to a set of triple templates $\mathcal{E}_\mathcal{Q}$ (and accordingly, a query set $\mathcal{Q}$) that are more semantically relevant to G (see "Query Transformer").

This step incurs a second round of communication between module $\mathcal{M}_{QX}$ and the LLM model $\mathcal{L}$. In this process, $\mathcal{M}_{QX}$ adopts a novel *Pattern-of-Thought* Prompting (PoT) that extends Chain-of-Thought (CoT) [8], which exploits the topology and connectivity of graph pattern queries to generate prompts (see "Prompt Generator").

$$\boxed{\begin{aligned} \mathcal{P}^i_2 &\leftarrow \mathcal{M}_{QX}(\mathcal{M}_P, Q^{I^i}) \\ \mathcal{E}^i_\mathcal{Q} &\leftarrow \mathcal{L}(\mathcal{P}^i_2) \end{aligned}}$$

Query Processing and Suggestion. GraphLingo then invokes a LLM-enhanced QueryProcessor Module $\mathcal{M}_S$ to process the queries and obtain both their answers A_{NL} and suggested queries Q'_{NL}. This completes the third round of communication, between the query processor $\mathcal{M}_s$ and $\mathcal{L}$.

$$\boxed{\begin{aligned} \mathcal{Q}^i(G) &\leftarrow \mathcal{M}_S(\mathcal{Q}^i)\text{(in parallel)} \\ \mathcal{L}(\mathcal{E}_\mathcal{Q}) &\leftarrow \mathcal{M}_s(\mathcal{M}_P, \mathcal{E}_\mathcal{Q}) \\ (A_{NL}, Q'_{NL}) &\leftarrow \mathcal{M}_s(\mathcal{Q}(G) \cup \mathcal{L}(\mathcal{E}_\mathcal{Q})) \end{aligned}}$$

Knowledge Augmentation. An (optional) post processing step is then performed to augment KG G with the statements from $\mathcal{L}(\mathcal{E}_Q)$ (via an augmentation operator $\oplus$). This step (characterized as $G^{i+1} = G^i \oplus \mathcal{L}(\mathcal{E}_\mathcal{Q})$) can be performed via additional validation by domain experts.

At this step, users can choose from a suggested query from Q'_{NL} or issue a new question to start the next session. Below we summarize the computation of GraphLingo.

Example 2: Consider a student exploring a curriculum KG G with an initial question Q_{NL}: "What professor is teaching Data Mining?" and prefers a more "conservative" exploration. GraphLingo responds in a single session as follows. (1) It exploits LLM (ChatGPT by default) with a prompt ($\mathcal{P}_1$) in Example 1, and translates Q_{NL} into a graph query Q^I with a single triple template $\mathcal{E}^I_Q$={(*?:Professor*, *teaches*, *Data Mining*)} (step 1). (2) Q^I is then further refined to a set of relevant queries $\mathcal{Q}$ (step 2). One top ranked query $\mathcal{Q}^i$ contains a path with two triple templates *(?:Professor, instructorOf, ?:Course)* and *(?:Course, hasTopic, Data Mining)* under *e.g.,* ontological edge to path transformation [9], which is more likely to have in-context matches in G. (3) GraphLingo then performs both graph search in G as well as an "open-ended" search with ChatGPT, and selects a top in-context

[1] Prompts are plain text; we underlined the structures for ease of reading.

```
Algorithm Q-Transformer
Input: Configuration $C = (\mathcal{E}_Q^I,\ \mathcal{T}_R,\ \mathcal{P}_2.\mathcal{T})$, threshold $k$
Output: a $\mathcal{P}_2$ prompt instance
queue $\mathcal{Q} := \emptyset$; prompt set $\mathcal{S}_\mathcal{P} := \emptyset$;
induced sub-queries set $\mathcal{E}_1 := \emptyset$;
for each $e_Q$ with variable node in $\mathcal{E}_Q^I$ do
    $e_Q$.weight := WUpdate($\mathcal{Q}$, $e_Q$, $u$, $v$);
    $\mathcal{Q} := \mathcal{Q} \cup e_Q$;
while $\mathcal{S}_\mathcal{P}$.size $< k$ and $\mathcal{Q} \neq \emptyset$ do
    $e := < u, r, v > :=$ $\mathcal{Q}$.dequeue();
    $\mathcal{E}_1 := \mathcal{E}_1 \cup e$;
    for each $e_Q$ in $e$.neighbors do
        if $e_Q$ not connected to $\mathcal{E}_1$ do
            $e_Q$.weight := WUpdate($\mathcal{Q}$, $\mathcal{E}_1$, $e_Q$);
            $\mathcal{Q} := \mathcal{Q} \cup e_Q$;
    $\mathcal{S}_\mathcal{P} := \mathcal{S}_\mathcal{P} \cup$ PoTGen($< u, r, v >$);
return $\mathcal{S}_\mathcal{P}$;

Procedure WUpdate($\mathcal{Q}, \mathcal{E}_1, e_Q$)
compute coverage with Equation 1;
compute and normalize cost with Equation 2;
compute relevancy with Equation 3;
return coverage + cost + relevancy;

Procedure PoTGen($< u, r, v >$)
prompt := $f_{\text{node}}(u)$ + $f_{\text{node}}(v)$;
prompt += $f_{\text{relation}}(r)$;
prompt += $f_{\text{triple}}(u, r, v)$;
prompt += $f_{\text{neighbors}}(u, r, v)$;
for each transformation function $T$ in $\mathcal{T}_R$ do
    transformed_triple := $T(< u, r, v >)$
    prompt += $f_{\text{example}}(T,$ transformed_triple$)$
return prompt;
```

Fig. 2: Algorithm Q-Transformer

answer as a path *(J.Ma, instructorOf, CSDS 435)*, *(CSDS 435, hasTopic, Data Mining)*. Meanwhile, it generates exploration questions by jointly diversifying triple templates from KG and LLM, such as an "in-context" question: "*What CSDS courses have the topic Knowledge Graph?*" and a more "open-ended" question "*What is the relationship between Data Mining and Machine Learning?*" for users to choose.

B. Modules and Algorithms

Prompt Generator ($\mathcal{M}_P$). This module plays a central role and is invoked by all other three modules. For communication steps (1) and (3), it follows the Prompt template $\mathcal{P}_1$ and $\mathcal{P}_3$ to generate corresponding prompts.

Prompt (Template) $\mathcal{P}_1$: **Examples** of pairs $\{Q_{NL}^i, \mathcal{E}_Q^I\}$ **Task**: Extrapolate a graph query equivalent to: (input NL query Q_{NL})

Prompt (Template) $\mathcal{P}_2$: **Context**: Description of each triple template $e_Q^I \in \mathcal{E}_Q^I$;
Description of neighbors of $e_Q^I \in G$;
Description of transformations (*e.g.,* ontology transformations);
Description of $\mathcal{E}_Q^I$'s constraint
Examples of identified transformation pairs
Task: Stepwise reasoning about context and exemplars; Give top-k transformed queries.

Prompt (template) $\mathcal{P}_3$: **Context**: Description of triples $e \in \mathcal{E}_Q$;
Task: Based on the given context, answer Q_{NL}.

For step (2) with input graph pattern Q^I as triple templates $\mathcal{E}_Q^I$, it follows a "Pattern-of-thought" (PoT) strategy (Algorithm 2), to best guide LLM to reasoning following *paths*. This mechanism enables LLM to comprehend the intricate relations between entities within the subgraphs, hence providing more "in-context" examples and prompts.

Algorithm. Given a configuration C including a set of triple templates $\mathcal{E}_Q^I$, a set of transformation $\mathcal{T}_R$, and the task of prompt template $\mathcal{P}_2.\mathcal{T}$, algorithm Q-Transformer generates a set of $\mathcal{P}_2$ prompt instances. Until reaching the bounded number of prompts, i.e. threshold k, at each iteration, Q-Transformer dynamically refines the set of weighted query triples, known as "pattern", by consulting two procedures WUpdate and PoTGen.

Auxiliary Structure. To estimate the weight of each triple to be in the pattern, we introduce three metrics: *query coverage*, *cost*, and *relevancy*. Given $\mathcal{E}$ as the original graph pattern query, $\mathcal{E}_1$ as decomposed sub-queries (sub-patterns) of $\mathcal{E}$, $|\mathcal{E}_i|$ is the node cardinality of any query, and e_Q as the candidate triple template to be added to $\mathcal{E}_1$

Query Coverage. The Coverage metric establishs a criteria of maximal coverage in term of equivalent query rewriting. We define **coverage** of the sub-patterns as:

$$\text{Coverage} = |\mathcal{E}| - |\mathcal{E}_1 \cup e_Q| \tag{1}$$

In the system of GraphLingo, the union cardinality of decomposed sub-patterns can be efficiently estmiated with HyperLogLog data structure.

Cost. We define the query **cost** of adding e_Q to it, or how efficient $\mathcal{E}_1$ processing is after taking e_Q, as:

$$\text{cost} = \prod_{i \in \mathcal{E}_1 \cup e_Q} \text{card}(i) \tag{2}$$

where i is the entities of the triple template within $\mathcal{E}_1 \cup e_Q$, and $\text{card}(i)$ is the cardinality function estimating how many instances that are equal to (or have the same type as) i if i is a real-world (or variable) entity, respectively.

Relevancy. We would like to quantify the relevancy of e_Q to $\mathcal{E}$, as well as the diversity of $\mathcal{E}_1^{'} = \mathcal{E}_1 \cup e_Q$:

$$\text{relevancy} = \frac{1-\lambda}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{sim}(e_Q, e) \tag{3}$$
$$+ \frac{\lambda}{|\mathcal{E}_1'|^2} \sum_{e_1 \in \mathcal{E}_1'} \sum_{e_2 \in \mathcal{E}_1'} [1 - \text{sim}(e_1, e_2)]$$

Descriptor. We then define a description function $f_x(c)$, where x can be among {node, relation, triple, neighbors, example}. $f_x(c)$ would take in an input c with corresponding type x and return a description of c as follows:

$$f_x(c) = \text{CONCAT}(c.\text{type}, \underset{\text{attr} \in c}{\text{CONCAT}}(\text{attr.name}, \text{attr.val})) \tag{4}$$

where attr is any pair of (attribute's name, attribute's value), and CONCAT is the string concatenation method.

Algorithm. Algorithm Q-Transformer (1) starts by prompting $\mathcal{L}$ to reason from a "start node" (a variable node by default) (line 3-5), and dynamically induces a set of weighted neighboring triples to be processed. At each iteration, algorithm Q-Transformer polls the triple template e with minimal weight from the priority queue $\mathcal{Q}$ and generates the corresponding transformation prompt instance of e: The weight $w(e_Q)$ for a triple template e_Q is estimated by a holistic aggregation of its processing cost, semantic closeness to $\mathcal{E}_Q^I$, and the amount of new information it may introduce if processed (Procedure WUpdate). (2) It follows a weighted spanning tree algorithm to dynamically induce the weighted triple templates as a sub-pattern (line 6-13), and use prompt template $\mathcal{P}_2$ to generate up to a bounded number of prompts (Procedure PoTGen).

(1) *Weight Updater*. Given a candidate triple template e to be added to a set of induced sub-patterns $\mathcal{E}_1$, and an original graph pattern query $\mathcal{E}_Q^I$, procedure WUpdate dynamically estimates the weight of each triple template based on three metrics: query coverage (Equation 1), query processing cost (Equation 2, and relevancy & diversity (Equation 3) of induced sub-patterns within a bounded error.

(2) *PoT Generator*. Given a triple template e, procedure PoTGen generates corresponding prompt instance of e based on the prompt template $\mathcal{P}_2$. Utilizing description functions $f_x(c)$ (Equation 4), it first describes each component, i.e. entity & relation, within the triple template, and an entire triple template as a whole. PoTGen then describe the neighborhoods of e following the same strategy. Finally, it transforms e based on a set of transformation functions, and describe each transformation as an identified example accordingly.

Example 3: Continuing with the task transforming $\mathcal{E}_Q^I$ = $\{$(*?:Professor*, *teaches*, *Data Mining*)$\}$. An instance prompt of $\mathcal{P}_2$, automatically generated by $\mathcal{M}_P$ is given as follows:

> **Context**: CSDS 435 is a 3-cred course: Data Mining. CSDS 435 has Topic: Data Mining, Topic:Cluster Analysis and (is) taught by Prof. J.Ma. Prof. J.Ma has research interest (of) Data Mining, Causal Inference. A Professor (is) an instructor of a Course that has a Topic. $\mathcal{E}_Q^I$ is finding a professor teaches Data Mining.
> **Examples**: Ontological transformation transforms a triple into a path with (possibly) new variable nodes based on the ontology of the knowledge graph. Ontological transformation indicates $\{$(*?:Professor*, *teaches*, *data mining*)$\}$ can transform to $\{$*(?:Professor, instructorOf,?:Course), (?:Course, hasTopic, Data Mining)*$\}$
> **Task:** Step-wise reasoning about each node, edge, and the relational structure given in context. Then based on that, continue to reason about provided examples. Finally, determine top-k best transformed queries.

Correctness. Algorithm Q-Transformer is guaranteed to terminate if either there is no triple template left to induce, or the prompts threshold k has been reached. Q-Transformer correctly induces a set of triple templates, as sub-patterns, that has minimal total weight. To prove that, we assume Q-Transformer does not return a (set of) minimum spanning tree. Then there must exist another minimum spanning tree T that has smaller weight than Q-Transformer's spanning tree. Let $e = (u, r, v)$ be the first triple template that is added by Q-Transformer, but not to T, and divide the minimum spanning tree into two cuts (X, Y). There should exist another triple $e' = (x, l, y)$ that is added to T in place of e connecting two cuts (X, Y). We know that priority queue $\mathcal{Q}$ maintains the triple template with minimal cost at top which is polled out (line 7) during each iteration, so e.weight $<$ e'.weight. Let $T' = T \setminus e' \cup e$. T' is still a tree, but T'.weight $<$ T.weight, which contradicts that T is a minimum spanning tree. □

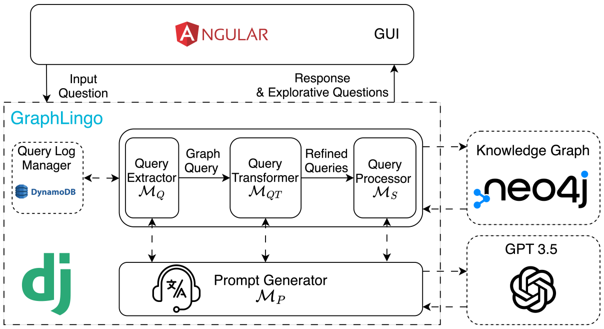

ANGULAR
GUI
Input Question
Response & Explorative Questions
GraphLingo
Query Log Manager
DynamoDB
Query Extractor $\mathcal{M}_Q$
Graph Query
Query Transformer $\mathcal{M}_{QT}$
Refined Queries
Query Processor $\mathcal{M}_S$
Knowledge Graph
neo4j
GPT 3.5
Prompt Generator $\mathcal{M}_P$
dj

Fig. 3: GraphLingo Architecture

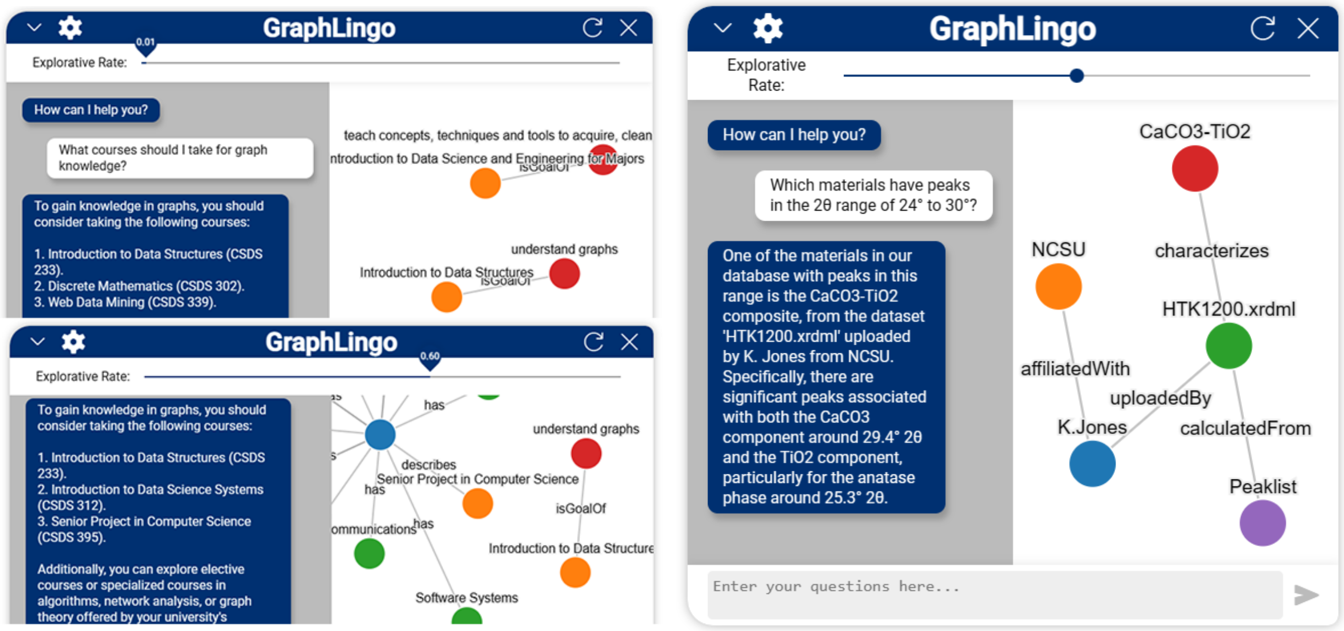
Time Cost. We note that each auxiliary structure takes constant processing time. Algorithm Q-Transformer considers up to $|\mathcal{E}_Q^I|$ triple templates. For each triple template, i.e. iteration, it takes WUpdate constant time to compute the associated weight, and also takes PoTGen constant time to generate triple description and transformation example with constant number of transformation function. It takes $O(log|\mathcal{E}_Q^I|)$ to maintain the priority queue per each iteration. Thus, the overall time cost of Q-Transformer is $O(|\mathcal{E}_Q^I|log|\mathcal{E}_Q^I|)$.

Query Transformer ($\mathcal{M}_{QT}$). This module maps Q^I to a query space that aligns better with G. It transforms each triple of Q^I by applying a predetermined set of transformation functions [9], then prompts LLM through the *Prompt Generator* with prompt $\mathcal{P}_2$ to step-by-step reason about each possible transformation and determine the best one.

LLM enhanced Query Processor ($\mathcal{M}_S$). This module applies parallel graph query processing to obtain the matched triples $\mathcal{Q}(G)$ over the query load $\mathcal{Q}$ (supported by graph databases). In parallel, given a graph pattern query $\mathcal{Q}^i$, a KG G, and a hyperparameter λ controlling explorative nature of the query processor, the module first quantify how relevant a triple in G is to $\mathcal{Q}^i$:

$$\text{rev}(t, \mathcal{Q}^i) = \frac{1}{|\mathcal{Q}^i|} \cdot \frac{E(t)}{E(G)} \cdot \sum_{q \in \mathcal{Q}^i} \text{sim}(t, q) \qquad (5)$$

where $\frac{E(t)}{E(G)}$ quantifies the popularity of t's relation type in G, and $\text{sim}(t, q)$ can be any similarity function, such as string distance, designed to quantify the semantic proximity between two triples. The module uses it to identify the top-k relevant triples from G, and subsequently determine the minimum spanning tree covering these triples. Finally, it prune the identified spanning tree to get $\mathcal{Q}^i(G)$ with both relevance and diversity maximized based on the following objective function:


GraphLingo
GraphLingo
GraphLingo
Enter your questions here...
teach concepts, techniques and tools to acquire, clean
ntroduction to Data Science and Engineering for Majors
isGoalOf
understand graphs
Introduction to Data Structures
isGoalOf
has
understand graphs
describes
Senior Project in Computer Science
has
isGoalOf
ommunications
has
Introduction to Data Structure
Software Systems
CaCO3-TiO2
NCSU
characterizes
HTK1200.xrdml
affiliatedWith
uploadedBy
K.Jones
calculatedFrom
Peaklist

Explorative Rate: 0.01

How can I help you?

What courses should I take for graph knowledge?

To gain knowledge in graphs, you should consider taking the following courses:

1. Introduction to Data Structures (CSDS 233).
2. Discrete Mathematics (CSDS 302).
3. Web Data Mining (CSDS 339).

Explorative Rate: 0.60

To gain knowledge in graphs, you should consider taking the following courses:

1. Introduction to Data Structures (CSDS 233).
2. Introduction to Data Science Systems (CSDS 312).
3. Senior Project in Computer Science (CSDS 395).

Additionally, you can explore elective courses or specialized courses in algorithms, network analysis, or graph theory offered by your university's

Explorative Rate:

How can I help you?

Which materials have peaks in the 2θ range of 24° to 30°?

One of the materials in our database with peaks in this range is the CaCO3-TiO2 composite, from the dataset 'HTK1200.xrdml' uploaded by K. Jones from NCSU. Specifically, there are significant peaks associated with both the CaCO3 component around 29.4° 2θ and the TiO2 component, particularly for the anatase phase around 25.3° 2θ.

(a) Academic KG Exploration: In-context vs. Exploratory

(b) Material Science KG Exploration

Fig. 4: GraphLingo Interface: Natural Language Q & A with Knowledge Graph View

$$\mathcal{L}(\mathcal{Q}^i(G)) = (1-\lambda) \sum_{e \in \mathcal{Q}^i(G)} \text{rev}(e, \mathcal{Q}^i) \quad (6)$$

$$+ \frac{\lambda}{|\mathcal{Q}^i(G)|} \sum_{e_1 \in \mathcal{Q}^i(G)} \sum_{e_2 \in \mathcal{Q}^i(G)} [1 - \text{sim}(e_1, e_2)]$$

Algorithm. (1) Selecting a subset of triple templates (or pruning) the original tree maximizing the objective function 6 is a NP-hard problem. Thus, we greedily choose a pair of triple templates (e_1, e_2) that maximizes Equation 6 until the objective function of the subset converges.

Meanwhile, (2) it treats LLM $\mathcal{L}$ as a "query processor", and directly requests to transform the extended queries (as triple templates $\mathcal{E}_\mathcal{Q}$) to a set of triple statements $\mathcal{L}(\mathcal{E}_\mathcal{Q})$ "in the wild" inherently from $\mathcal{L}$'s external knowledge. GraphLingo then performs a *diversified* selection over the union $\mathcal{Q}_{G,\mathcal{L}} = \mathcal{Q}(G) \cup \mathcal{L}(\mathcal{E}_\mathcal{Q})$ that conforms to user's preference on "exploratory", by maximizing a bi-criteria function $(1-\lambda)\sum_{e\in\mathcal{Q}(G)} \text{rev}(e, \mathcal{Q}) + \lambda \sum_{e\in\mathcal{L}(\mathcal{E}_\mathcal{Q})} \text{sim}(e, \mathcal{Q})$. Here function rev (resp. sim) quantifies the semantic closeness measure of $\mathcal{Q}(G)$ (resp. similarity of the representations of those found by LLM on Web) with $\mathcal{Q}$.

Time Cost We note that both (1) and (2) are MaxSumDispersion problems [10] that can be solved with a greedy approach taking $O(nlog(n))$ time, where $n = |\mathcal{Q}^i|$ for (1) and $n = |\mathcal{Q}_{G,\mathcal{L}}|$ for (2). It also takes procedure (2) $O(I_{\text{LLM}})$ to request an answer from LLM $\mathcal{L}$, where I_{LLM} denotes the average time to request an answer from $\mathcal{L}$. Thus, the overall time cost is $O(|\mathcal{Q}_{G,\mathcal{L}}|log|\mathcal{Q}_{G,\mathcal{L}}| + I_{\text{LLM}})$.

C. Workflow Time Analysis

We denote the average time it takes to request an answer from LLM as I_{LLM}. In total, there are three rounds of communication between GraphLingo and LLM $\mathcal{L}$. Prompt generation for $\mathcal{P}_1$ and $\mathcal{P}_3$ takes time $O(|Q|)$, while inducing the pattern template in $\mathcal{P}_2$ requires $O(|\mathcal{E}_Q^I| \log |\mathcal{E}_Q^I|$ with the Q-Transformer algorithm. Meanwhile, the overall time cost of query processing algorithm $\mathcal{M}_S$ is $O(|\mathcal{Q}_{G,\mathcal{L}}|log|\mathcal{Q}_{G,\mathcal{L}}| + I_{\text{LLM}})$. Thus the total cost is in $O(I_{\text{LLM}} + |Q| + |\mathcal{E}_Q^I| \log |\mathcal{E}_Q^I| + |\mathcal{Q}_{G,\mathcal{L}}|log|\mathcal{Q}_{G,\mathcal{L}}|)$ time. In practice we observe that I_{LLM}, the online communication cost is often dominating the local processing time $O(|Q| + |\mathcal{E}_Q^I| \log |\mathcal{E}_Q^I| + |\mathcal{Q}_{G,\mathcal{L}}|log|\mathcal{Q}_{G,\mathcal{L}}|)$, hence each session incurs a dominating cost $O(I_{\text{LLM}})$.

III. Architecture

GraphLingo is built upon a three-tier architecture, as illustrated in Fig.3 and conceptually presented in Figure 1. (1) Users engage GraphLingo through an interactive GUI that is built in Angular framework, extended to support an interactive visual knowledge graph (KG) panel (Fig. 4). (2) At the core of GraphLingo are graph exploration modules deployed using Django and Docker, serving as a bridge between pluggable knowledge graph and a Language Model (LLM). The KG is stored using Neo4j graph database. We adopt GPT 3.5 in our demo. Hugging Face models are utilized to furnish encoded similarity measurements within the modules. (3) We also optimize the system by storing session history in DynamoDB. This not only helps accelerating query processing by accessing past queries views, but also allows a "rewind" to re-explore from a certain timestamp. We report the details in [7].

IV. Demonstration Overview

Set up. We demonstrate GraphLingo through an interactive Q & A session, navigating a real-world, specialized academic information KG from the Department of Computer and Data Sciences at Case Western Reserve University. This KG encompasses diverse academic information, including details about courses, professors, and degrees. We also showcase application of GraphLingo in material science search with a specialized XRD data, scripts and workflows KG from CRUX [1].

NL-based Domain-Knowledge Exploration. We invite users to explore the academic KG through the interactive $Q\&A$ interface of GraphLingo. Users can effortlessly input their exploration queries in natural language. GraphLingo subsequently processes the queries automatically through a series of continuous interactions with the KG and LLM. The results are then presented in natural language for novice users, while professional users can engage with the visual graph interface to gain a more nuanced understanding for exploration.

Exploring with preference tuning. Users may not be satisfied with the provided answers that are either too "constrained" or too "open-end". We invite users to experience the ad-hoc tuning of GraphLingo to let it output answers and suggest questions towards more desired preference, with a simple "slide and play" action, and let them observe the difference.

Example 4: Figure 4a showcases preference tuning. (1) With a more conservative manner, i.e. $\lambda = 0.01$, GraphLingo prefers "in-context" answer with courses directly related to 'graph data' like Data Structures or Web Data Mining. A more explorative preference, where $\lambda = 0.6$, included Data Science System and Senior Project, which involve 'graph data' in a different perspective, such as subtopics including graph visualization, based on inferred context from the KG.

Online academic advising. In this scenario, we invite users to experience how GraphLingo supports a new data science education program as a chatbot application. Knowing little about the syllabus of the many newly opened courses, students start with vague questions such as "what to learn to know more about AI?" GraphLingo will guide them through course information, study plan and advisor information, among others, by exploring curriculum knowledge graph enhanced with GPT. This helps students retrieve desired knowledge, ranging from detailed courses to more open-end questions.

Material Scientific Data Search. In our second scenario, we invite users to explore factual materials science knowledge from X-ray diffraction (XRD) data analysis, and in particular, peak analysis. Supported by a crowd-sourced domain-specific materials knowledge graph (CRUX) [1], GraphLingo will support more in-context questions to help material data scientists find the datasets, filtering specific peak locations, and experimental setting (*e.g.,* temperature ranges), or help general public to obtain common knowledge by LLM from Web.

References

[1] M. Wang, H. Ma, A. Daundkar, S. Guan, Y. Bian, A. Sehirlioglu, and Y. Wu, "Crux: Crowdsourced materials science resource and workflow exploration," in *CIKM*, 2022.

[2] Y. Li, V. Zakhozhyi, D. Zhu, and L. J. Salazar, "Domain specific knowledge graphs as a service to the public," in *KDD*, 2020.

[3] OpenAI, "Gpt-4 technical report," in *ArXiv*, 2023.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, and et al., "Palm: Scaling language modeling with pathways," in *ArXiv*, 2022.

[5] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," in *ArXiv*, 2023.

[6] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," in *TKDE*, 2015.

[7] D. Le, K. Zhao, M. Wang, and Y. Wu, "Graphlingo(full version)," 2023. [Online]. Available: https://github.com/Escanord/GraphLingo

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, 2022.

[9] S. Yang, Y. Wu, H. Sun, and X. Yan, "Schemaless and structureless graph querying," in *PVLDB*, 2014.

[10] R. Hassin, S. Rubinstein, and A. Tamie, "Approximation algorithms for maximum dispersion," in *Operations Research Letters*, 1997.