An Interactive Visual Paradigm for Knowledge Graph Question-Answering

by

Vayd Sai Ramkumar

S.B. in Artificial Intelligence and Decision-Making and in Mathematics, MIT, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by:	Vayd Sai Ramkumar Department of Electrical Engineering and Computer Science May 9, 2025
Certified by:	Lalana Kagal Principal Research Scientist, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

An Interactive Visual Paradigm for Knowledge Graph Question-Answering

by

Vayd Sai Ramkumar

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

In an era of information overload, verifying data reliability and provenance is critical, yet knowledge graphs (KGs) often remain complex for non-expert users. This thesis introduces TRACE, a Reasoning and Answer-path Comprehension Engine, a visualization tool enhancing transparency in KG question answering (KGQA). By abstracting intricate KGs into intuitive meta-nodes, TRACE simplifies exploration of large, multi-topic datasets. Its interactive interface allows users to navigate semantic communities and trace reasoning paths, fostering trust through clear answer derivation. Unlike cluttered traditional graph visualizations, TRACE's meta-node approach provides a scalable, user-friendly solution, concealing technical complexities while enabling robust query validation. Large language models support natural language query parsing and community summarization, making KGs accessible to diverse audiences. TRACE positions itself as a vital widget for information platforms, empowering users to counter misinformation confidently. A user study and pipeline evaluation confirmed TRACE's intuitive interface excels for complex queries, though multi-hop paths pose challenges, while processing tests demonstrated its scalable paradigm for large datasets. By prioritizing transparency and usability, TRACE redefines KGs as reliable tools for knowledge discovery, laying a foundation for future systems to deliver trustworthy, accessible information in a digital landscape fraught with uncertainty.

Thesis supervisor: Lalana Kagal
Title: Principal Research Scientist

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The digital era has ushered in an unprecedented surge of information, where the relentless flow of data through social media, online platforms, and digital archives has fundamentally reshaped how society engages with knowledge. This age of information abundance, while rich with opportunity, presents profound challenges: individuals are inundated with content, struggling to separate truth from falsehood amid a cacophony of competing narratives. The societal imperative to navigate this complexity with clarity and confidence has never been more urgent, as misinformation proliferates and trust in information sources erodes. From health crises fueled by false claims to public debates swayed by distorted facts, the stakes of unverified information are high, demanding tools that not only deliver answers but also reveal their origins and logical foundations. This thesis introduces TRACE, a Reasoning and Answer-path Comprehension Engine, as a pioneering response to these challenges, leveraging the structured power of knowledge graphs (KGs) to create an intuitive, transparent interface that empowers users to explore and verify information in a digital landscape fraught with uncertainty.

The genesis of this research lies in the recognition that existing information access systems often exacerbate rather than alleviate the burdens of information overload. Traditional search engines, while adept at retrieving vast arrays of data, frequently produce results that lack context or traceability, leaving users to navigate a maze of fragmented or contradictory information. Similarly, text-based question-answering systems, reliant on unstructured data, risk delivering responses that are compelling yet unverified, amplifying the potential for misinformation. These shortcomings are particularly acute in contexts where precision and trust are paramount—scientific inquiry, medical decision-making, or civic engagement—where erroneous information can lead to significant harm. Knowledge graphs offer a compelling alternative, organizing information as interconnected entities and relations that enable precise, contextually grounded queries. By structuring facts as relational networks—linking

researchers to their institutions or diseases to their treatments—KGs facilitate reasoning that is both logical and verifiable. Yet, their inherent complexity, often encompassing millions of connections, renders them daunting for non-expert users, who lack the technical acumen to engage with intricate data structures or formulate sophisticated queries. This accessibility gap, set against the backdrop of societal demands for reliable information, drives the development of TRACE, which seeks to transform how users interact with KGs through a groundbreaking visualization approach.

At the heart of this research is a response to the societal scourge of misinformation, which thrives in environments overwhelmed by data. The rapid spread of false narratives—whether about public health, elections, or scientific findings—underscores the need for systems that anchor information in verifiable structures. Knowledge graphs, with their ability to encode validated facts and relationships, provide a robust foundation for countering misinformation by ensuring answers are drawn from explicit, traceable data. However, unlocking this potential requires interfaces that make KGs intuitive and approachable, particularly for those without specialized training. TRACE addresses this by reimagining KG exploration through a visual paradigm that distills complex networks into cohesive, user-friendly representations. By grouping related entities into navigable clusters, TRACE enables users to explore diverse topics—from literature to biomedicine—without being overwhelmed by technical complexity. More critically, it allows users to visually trace the logical paths behind answers, offering a clear audit trail that reveals how conclusions are reached, such as connecting a researcher's work to a high-impact journal through institutional affiliations. This transparency is not merely a technical innovation but a societal necessity, empowering users to assess the reliability of information in a way that fosters trust and counters skepticism.

The motivation for TRACE also stems from the evolving interplay between technology and human understanding. As artificial intelligence, particularly large language models (LLMs), becomes ubiquitous in information systems, there is an urgent need to balance their interpretive flexibility with structured rigor. LLMs excel at processing natural language, making them powerful tools for query formulation, but their tendency to generate unverified or speculative responses poses risks in high-stakes contexts. Knowledge graphs provide a complementary framework, grounding LLM outputs in formal, structured data to ensure accuracy. TRACE harnesses this synergy, integrating LLMs to simplify query input and summarize content while anchoring answers in KG relationships, creating a seamless bridge between human inquiry and computational precision. This approach reflects a broader societal shift toward technologies that enhance critical thinking, enabling users to engage actively with information rather than passively accepting it.

Inclusivity is another cornerstone of this research, driven by the need to democratize access

to complex data systems. Knowledge graphs, traditionally the purview of data scientists, exclude non-expert users—educators, journalists, or policymakers—who could benefit from their insights. This exclusion limits the societal impact of KGs, particularly in an era where informed decision-making is essential across all sectors. TRACE's visualization paradigm prioritizes accessibility, offering an interface that requires no prior knowledge of graph theory or query languages. By enabling users to explore thematic clusters and generate questions effortlessly, TRACE ensures that diverse audiences can harness the power of KGs, fostering a more equitable knowledge ecosystem. This inclusivity aligns with the broader goal of empowering individuals to navigate the information landscape with agency and confidence.

Finally, this research is motivated by the growing demand for transparency in automated systems. As society grapples with opaque technologies, there is a pressing need for tools that demystify decision-making processes, particularly in contexts where trust is critical. TRACE's ability to visualize reasoning paths—showing, for example, how a query about a historical event connects to primary sources—meets this need by making the derivation of answers explicit. By envisioning TRACE as an integrative component for broader platforms, such as search engines or educational tools, this work lays the groundwork for a future where transparency and reliability are intrinsic to information access. In addressing the societal challenges of information overload, misinformation, and inaccessibility, TRACE offers a novel approach to KGQA, redefining how users engage with knowledge and setting a foundation for trustworthy, user-centric systems.

1.1 Background

Knowledge graphs (KGs) constitute a sophisticated class of graph-based data structures designed to encapsulate explicit semantics through a formal ontology that delineates entity types, permissible relations, and their hierarchical interdependencies, as comprehensively outlined by Hogan et al. [1]. Unlike conventional graphs, which are restricted to representing nodes and edges devoid of intrinsic meaning—such as a mere connection between `Paris` and `France`—KGs enrich their structure with domain-specific significance. For example, a KG explicitly asserts the triplet (`Paris`, `is_capital_of`, `France`), where `Paris` is typed as *city* and `France` as *country*, enabling semantically grounded reasoning and context-aware querying. This semantic richness underpins a wide array of applications, including search engines, recommendation systems, and advanced question answering frameworks. Modern KGs leverage entity embeddings, which are vector representations in a high-dimensional space $\mathbb{R}^d$, to numerically capture these semantics, facilitating operations such as similarity-based retrieval via cosine similarity. Additionally, large language models (LLMs) enhance the

accessibility and interpretability of KGs by generating or parsing natural language descriptions aligned with their structured content.

1.1.1 Core Components

At the heart of a KG lies the triplet, formally denoted as (s, p, o), where $s \in \mathcal{E}$ represents the subject entity, $p \in \mathcal{R}$ the predicate or relation, and $o \in \mathcal{E}$ the object entity, with $\mathcal{E}$ and $\mathcal{R}$ being the sets of entities and relations, respectively. An illustrative triplet, $(\texttt{Paris}, \texttt{is\_capital\_of}, \texttt{France})$, encodes a factual assertion within the KG. This foundational structure is augmented by several critical components that collectively enhance its expressive power and utility in structured data representation.

Entities within a KG are endowed with types drawn from a predefined set $\mathcal{T}$, such that each entity $e \in \mathcal{E}$ is assigned a type $\tau(e)$—for instance, $\tau(\texttt{Paris}) = \textit{city}$. These type assignments impose constraints on permissible relations, ensuring that only entities of specific types may participate in certain predicates (e.g., only entities of type *person* may engage in `worksFor`). Relations themselves may exhibit hierarchical structures, wherein a sub-relation $p' \subseteq p$ inherits properties from its parent relation p. For example, `is_capital_of` may be a specialized subset of `is_located_in`, allowing the inference of broader geographical facts from specific assertions through deductive reasoning processes.

Further enriching this framework, entities are annotated with attributes, represented as key-value pairs $a : v$, such as $\texttt{Person\_X} : \texttt{birthYear} = 1990$ or $\texttt{Article\_Y} : \texttt{publicationDate} = 2023$. These attributes provide immediate access to entity-specific data without necessitating additional relational edges, thereby streamlining queries that target specific properties. To capture the latent semantics of these attributes and entities, embeddings are employed, mapping each entity e and its attributes into a vector $\mathbf{e} \in \mathbb{R}^d$. Models like Sentence-BERT [2] can generate such embeddings, enabling the computation of similarities between entities via cosine similarity, $\cos(\mathbf{e}_i, \mathbf{e}_j)$, which quantifies the alignment of their vector representations. For instance, this technique can identify clusters of individuals born in proximate years or articles published within similar timeframes.

The ontology or schema, typically formalized using standards like RDF Schema or OWL [3], serves as the backbone of the KG's semantic consistency. It specifies the domain and range of each relation—e.g., $\texttt{worksFor} : \textit{person} \rightarrow \textit{organization}$—and enforces type constraints that prevent erroneous triplet formations. This structured ontology enables type-constrained queries such as retrieving all researchers affiliated with universities in a specific city by traversing typed relations and inspecting associated attributes. To optimize such operations over large-scale KGs, graph partitioning algorithms like METIS [4] can be applied, dividing

the KG into semantically coherent subgraphs. This partitioning reduces computational overhead by localizing query traversals to relevant clusters, enhancing efficiency in real-world applications.

1.1.2 Representation Models

The practical instantiation of KGs relies on two predominant representation models, each tailored to specific storage, querying, and scalability requirements, thereby supporting the diverse needs of knowledge-based systems.

The Resource Description Framework (RDF) organizes KG data into triples, which are stored in specialized databases known as triple stores and queried using the SPARQL protocol [3]. RDF's standardized format ensures interoperability across heterogeneous datasets, as exemplified by large-scale KGs like DBpedia or Wikidata, which facilitate cross-domain queries spanning topics from geography to biomedicine. The enforcement of a schema in RDF guarantees data integrity by restricting triplet assertions to those compliant with the ontology, such as preventing a *city* from being the subject of `worksFor`. Query optimization in RDF systems can leverage METIS to partition the underlying graph into manageable substructures, minimizing the computational cost of SPARQL execution over massive datasets by focusing traversals on pertinent subgraphs. Furthermore, embeddings of RDF entities and relations can accelerate operations like similarity-based joins or semantic filtering, precomputing vector representations that align with the KG's structured content.

In contrast, labeled property graphs offer a more flexible paradigm, wherein both nodes (entities) and edges (relations) may carry attributes, as implemented in platforms like Neo4j and queried using languages such as Cypher. This model excels in applications requiring direct manipulation of properties, such as social network analysis, where one might query for all tech companies with CEOs under 40 by inspecting node attributes like `CEO_age`. The embeddings of these attributes can be clustered using cosine similarity to reveal latent patterns—e.g., grouping researchers by shared expertise or companies by industry focus—enhancing the KG's analytical capabilities. The adaptability of property graphs supports rapid prototyping and schema evolution, making them ideal for dynamic environments where data requirements shift over time.

Both RDF and property graph models preserve the semantic precision necessary for sophisticated queries, leveraging their typed relations and attributes to deliver contextually relevant results. Large language models can further augment these systems by translating natural language inputs into formal queries, aligning user intent with the underlying KG structure and broadening accessibility for non-technical users.

1.1.3 Extensions and Ecosystem

To maximize their functionality and adaptability, KGs are embedded within a rich ecosystem of extensions that enhance their reasoning, integration, and maintenance capabilities, catering to the demands of real-world applications.

Inference engines constitute a pivotal extension, employing logical rules to derive implicit facts from explicit triplets. For instance, given the triplet (`Alice`, `worksFor`, `Company_X`) and a rule defining `employs` as the inverse of `worksFor`, the engine infers (`Company_X`, `employs`, `Alice`). This deductive capability ensures logical completeness and consistency across the KG. Embeddings can validate such inferences by ensuring that vector representations of inverse relations exhibit high cosine similarity, reinforcing the semantic coherence of the inferred facts and enabling similarity-based consistency checks.

Alignment tools address the challenge of entity reconciliation across disparate datasets, a critical task in federated or collaborative KGs. For example, distinct references like `Company_X` and `X_Corp` may be merged into a single entity by comparing their attributes and relations. This process is significantly enhanced by embeddings, where entities with proximate vector representations—quantified via $\cos(\mathbf{e}_i, \mathbf{e}_j)$—are clustered to identify probable matches, reducing manual effort and improving data integrity. Such alignment is indispensable for integrating heterogeneous sources into a unified knowledge base.

Versioning and provenance tracking mechanisms record the evolution of the KG, capturing changes such as triplet additions or modifications alongside their origins—e.g., `Source_Y` asserting (`Article_B`, `publishedIn`, `Journal_A`) on 2022-01-15. This temporal and source metadata enables queries over historical states or validation of data lineage, which is particularly valuable in domains like scientific research or legal documentation. LLMs can enhance this functionality by generating natural language summaries of provenance, such as "This fact was sourced from Journal_A's 2022 archive," improving transparency and usability for end users.

Collectively, these extensions transform KGs into dynamic, scalable systems, where embeddings optimize similarity-based operations, METIS enhances computational efficiency through partitioning, and LLMs bridge the gap between structured data and human interaction.

1.1.4 What Makes a Knowledge Graph

The distinction between a standard graph and a KG lies in the latter's semantic overlay, which elevates it beyond a mere topological structure defined as $G = (V, E)$, with V as vertices and $E \subseteq V \times V$ as edges. In a KG, nodes and edges are assigned interpretable identifiers—entities and relations, respectively—governed by an ontology that aligns with human-understandable

concepts, per Hogan et al. [1]. For instance, a KG might include entities like `Professor`, relations like `Teaches`, and objects like `Course`, with constraints ensuring that only professors teach courses. This semantic layer enables conceptual queries—e.g., identifying all drugs targeting a specific disease—that demand an understanding of relational semantics and entity types, rather than mere adjacency or connectivity.

The power of this structure is amplified by advanced computational techniques. METIS can partition a KG into semantically meaningful communities, such as academic versus industrial subgraphs, optimizing query execution by isolating relevant regions of the graph. Embeddings further enhance this capability by capturing semantic relationships in vector space, allowing cosine similarity to measure the relatedness of entities or relations—e.g., grouping diseases with similar treatment profiles. LLMs contribute by interpreting ambiguous or natural language queries, mapping them onto the KG's formal ontology to deliver precise, contextually appropriate responses.

1.1.5 Example: Codifying a Small Knowledge Graph

A small academic knowledge graph (KG) illustrates its construction and utility. The KG comprises entities, relations, and triplets, defined as follows:

- **Entities**: {Dr_Smith, Dr_Jones, Uni_XYZ, Journal_A, Article_B, Article_C}.
- **Relations**: {worksAt, wrote, publishedIn, coauthors, cites}.
- **Triplets**:
  - (Dr_Smith, worksAt, Uni_XYZ),
  - (Dr_Smith, wrote, Article_B),
  - (Article_B, publishedIn, Journal_A),
  - (Dr_Jones, coauthors, Dr_Smith),
  - (Dr_Jones, wrote, Article_C),
  - (Article_C, cites, Article_B).

Attributes add detail:

- Dr_Smith: expertise = AI,
- Uni_XYZ: city = London,
- Article_B: year = 2023,

- Journal_A: impactFactor = 5.2.

This structure is shown in Figure 1.1.

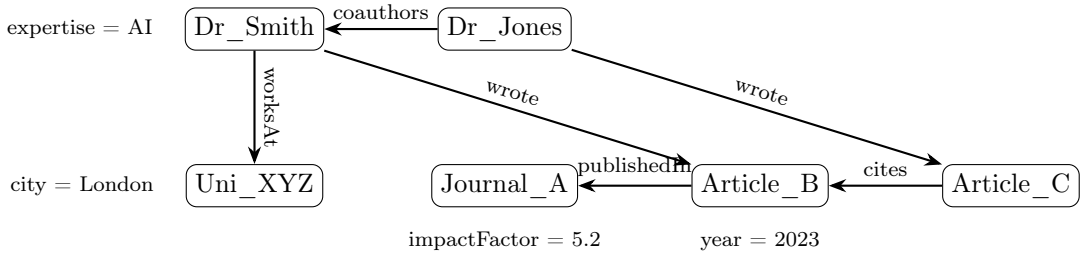


Figure 1.1: Academic knowledge graph with entities, relations, and attributes.

The schema ensures type constraints, e.g., `publishedIn` applies only to articles and journals, preventing invalid triplets like (Dr_Smith, publishedIn, Journal_A). A query, "Find London-based researchers who coauthored articles in high-impact journals," requires multi-hop traversal (see Figure 1.1): from Uni_XYZ (city = London) via worksAt to Dr_Smith, then coauthors to Dr_Jones, and via wrote and publishedIn to Journal_A (high impactFactor). This shows the KG's structured reasoning capability.

Embeddings can cluster researchers by expertise (e.g., Dr_Smith with AI experts) using cosine similarity. Large language models can reformulate natural language queries into precise traversal paths, enhancing accuracy and usability.

1.1.6 Question Answering (QA)

Knowledge graph question answering (KGQA) leverages the structured nature of KGs to automate the retrieval or inference of answers, distinguishing it from traditional QA systems that operate over unstructured text [5]. For example, the query "What is the capital of Brazil?" is resolved by retrieving the triplet (`Brazil`, `has_capital`, `Brasília`), delivering a precise, traceable response. The KGQA process encompasses several stages: parsing a natural language query into a formal representation (e.g., SPARQL), traversing the KG to locate relevant entities and relations, and ranking potential answers when multiple candidates emerge.

Single-hop queries involve direct lookups, such as "Where does `Dr_Smith` work?" which translates to `SELECT ?place WHERE { Dr_Smith worksAt ?place }`, yielding `Uni_XYZ`. Formally, this retrieves o where $(s, p, o) \in \mathcal{T}$, with $\mathcal{T}$ as the triplet set. Multi-hop queries, such as "Which city hosts `Dr_Smith`'s university?" require sequential traversal—e.g., `Dr_Smith` $\xrightarrow{\texttt{worksAt}}$ `Uni_XYZ` $\xrightarrow{\texttt{city}}$ `London`—expressed as finding a path $s \xrightarrow{p_1} e_1 \xrightarrow{p_2} e_2$. METIS can optimize such

traversals by partitioning the KG into localized subgraphs, reducing the search space for multi-hop operations.

Challenges in KGQA include entity disambiguation (e.g., distinguishing between multiple entities named `Paris`)), handling vague queries, bridging lexical gaps between user input and KG terminology, ensuring scalability over large graphs, and providing interpretable results. Embeddings address these by enabling similarity-based ranking—e.g., prioritizing answers with higher $\cos(\mathbf{e}_i, \mathbf{q})$ relative to a query vector $\mathbf{q}$—while LLMs enhance parsing and explanation, generating responses like "Dr_Smith works at Uni_XYZ, located in London." Together, these techniques ensure robust, efficient, and user-friendly question answering over KGs.

Chapter 2

Related Work

Knowledge graphs (KGs) have become an influential paradigm for structuring and querying information in diverse domains, ranging from scholarly data to commercial knowledge bases. By representing entities as nodes and relations as edges, KGs provide semantic and topological clarity that can unlock advanced query capabilities, facilitate exploratory analysis, and support question-answering workflows. This section surveys visual KG systems, examines prompting strategies for large language model (LLM)-based reasoning, discusses data visualization and community detection, and outlines key methods for mitigating hallucinations. Together, these areas form the basis for designing user-focused knowledge graph applications that emphasize transparency, interpretability, and robust query responses.

2.1 Fundamentals of Knowledge Graphs

Knowledge graphs structure information as a network of nodes, representing entities, and edges, denoting relations, typically expressed as triplets in the form (subject, predicate, object). The predicate in each triplet defines the semantic relationship connecting the subject to the object, enabling the representation of diverse knowledge domains [1]. For example, a triplet might capture a straightforward fact, such as *Paris* $\rightarrow$ *is capital of* $\rightarrow$ *France*, or a specialized relationship, such as *Protein_ X* $\rightarrow$ *inhibits* $\rightarrow$ *Pathway_ Y*. This triplet-based model provides a versatile framework for encoding relational networks, making knowledge graphs a robust tool for organizing and querying structured information across varied contexts.

The scalability and heterogeneity of knowledge graphs are critical attributes that shape their implementation and utility. Knowledge graphs span a wide range of sizes, from small, curated datasets with hundreds of triplets to massive repositories containing billions of triplets, such as Wikidata or the Microsoft Academic Graph. Large-scale graphs require advanced infrastructure, often utilizing specialized graph databases like Neo4j or distributed

systems to handle intensive query demands efficiently [6]. Smaller graphs, by contrast, can be managed using lightweight tools like `NetworkX`, which support rapid prototyping and iterative experimentation. Heterogeneity manifests in the diverse entity and relation types integrated within a single graph, facilitating cross-domain analysis. For instance, a knowledge graph might interlink geographical locations, biological molecules, and historical figures, each associated with distinct relation types. This diversity enhances the graph's expressive power but necessitates well-defined schemas or ontologies to maintain semantic consistency across heterogeneous data, ensuring that queries yield coherent and accurate results.

Knowledge graphs enable a broad spectrum of applications by supporting multi-hop reasoning and rich contextual analysis, which are essential for tasks requiring deep relational insights. In semantic search, knowledge graphs enhance query accuracy by providing contextual associations. A query for *Berlin* might return not only direct information about the city but also related entities like *Germany's capital* or *Berlin Wall*, enriching search outcomes. Recommender systems utilize knowledge graphs to generate tailored suggestions by modeling relationships among users, items, and contextual features [7]. For example, a music streaming service might leverage a graph to connect users to songs via genres, artists, and listening history, enabling recommendations that capture subtle preferences. In question answering, knowledge graphs support precise responses by facilitating logical traversals across entities and relations. A query such as *Who is the sibling of the author of Pride and Prejudice?* can be resolved by navigating from *Pride and Prejudice* to its author, Jane Austen, and then to her siblings, ensuring answers are grounded in explicit paths. In specialized domains, knowledge graphs empower researchers to uncover insights beyond the capabilities of traditional databases. In biomedicine, graphs linking genes, diseases, and treatments enable the discovery of novel therapeutic pathways. In finance, graphs connecting companies, transactions, and market trends support fraud detection and risk analysis. In academia, graphs like the Microsoft Academic Graph interlink publications, authors, and citations, facilitating studies of research influence and interdisciplinary connections. These applications underscore the power of knowledge graphs to address relation-driven challenges across diverse fields.

Query performance in knowledge graphs hinges on several technical considerations, including indexing strategies, query languages, caching mechanisms, and the nature of the query. Simple queries, such as retrieving all entities with a specific attribute, benefit from basic indexing and execute quickly. However, complex queries requiring multi-hop traversals, such as identifying paths across multiple relations, demand sophisticated heuristics or optimized indexing to maintain efficiency [8]. Query languages like SPARQL for RDF graphs and Cypher for property graphs offer expressive frameworks for defining these traversals, though

their performance depends on the graph's scale and structure. In interactive settings, where low-latency responses are critical, developers optimize storage layouts, employ distributed computing, or precompute common paths to minimize delays. As knowledge graphs grow, balancing the expressiveness of queries with computational scalability becomes a pivotal design challenge, requiring careful trade-offs to ensure both flexibility and responsiveness.

Standardized frameworks enhance the interoperability and semantic depth of knowledge graphs. The Resource Description Framework (RDF) provides a universal model for encoding triplets, enabling seamless integration of disparate data sources. The Web Ontology Language (OWL) extends RDF by defining rich schemas and logical constraints, supporting advanced reasoning over data [3]. Alternatively, labeled property graphs, as used in Neo4j, store attributes directly on nodes and edges, streamlining data manipulation for enterprise applications. These frameworks shape how developers define constraints, perform reasoning, and build user interfaces for querying. RDF-based graphs excel in open-domain scenarios requiring broad compatibility, while property graphs are optimized for rapid, attribute-driven queries. The choice of framework influences the graph's structure, query capabilities, and visualization strategies, impacting the overall system design.

Knowledge graphs are defined by their adaptable structures, explicit semantics, and ability to model intricate relationships. Their capacity to scale across domains and support applications like semantic search, recommendation, question answering, and domain analytics positions them as a cornerstone of modern data systems. The expanding ecosystem of query engines, visualization tools, and language model-driven reasoning methods highlights their transformative potential for knowledge organization and access.

2.2 Visual Knowledge Graph Systems

Although graph visualization libraries are plentiful, only a subset specifically target user-friendly knowledge graph exploration and question answering. Two well-known systems—**KG4Vis** and **LinkQ**—offer different approaches to bridging users and KGs, emphasizing interactive features and iterative query logic. Additionally, systems like **QAnswer** cater to domain-specific needs, while gaps remain in providing fully transparent subgraph and reasoning path visualizations.

KG4Vis.

KG4Vis [9] learns embeddings for data features, data columns, and prospective visualization designs, forming a knowledge graph of visualization choices and data properties. It recommends suitable visual encodings by comparing newly encountered data with an existing

embedding space. This system is valuable in data science workflows where efficient selection of charts or plots is critical, yet it lacks robust capabilities for subgraph retrieval or multi-hop reasoning paths. Its emphasis rests on guiding visualization decisions, rather than supporting knowledge graph question answering at a finer granularity.

LinkQ.

In contrast, LinkQ [10] dynamically transforms free-form user questions into a structured graph query, refining the query iteratively when the system detects ambiguities. This iterative disambiguation process is grounded by the graph's existing content, reducing off-topic outputs. User studies indicate that LinkQ supports exploratory data tasks by clarifying how queries map to the graph. However, LinkQ does not inherently display the subgraph or chain-of-thought behind the final answer. Users see query transformations but lack a broad graph-level perspective, making it harder to contextualize how distant entities might be related.

QAnswer and Domain-Specific Systems.

Other systems, such as QAnswer [11], focus on domain-specific question answering, often relying on standard graph queries and assuming a well-defined KG that users query directly. These systems simplify knowledge exploration in specialized contexts, such as biomedical or encyclopedic data, but typically do not prioritize advanced visualization of relevant subgraphs or elaborate reasoning paths. Consequently, while they are effective for targeted use cases, they do not fully address the need for transparent, user-oriented interfaces that illuminate the derivation of answers from the underlying graph structure.

Visualization Rationale.

The importance of visualization in KG systems cannot be overstated, as it plays a pivotal role in making graph structures accessible and understandable, particularly for non-expert users. By presenting KGs in a visual format, systems can hide technical complexities, such as SPARQL queries and RDF structures [10], allowing users to focus on the meaning of the data rather than its underlying representation. Interactive visualizations enable iterative exploration, where users can expand nodes, filter information, and navigate through the graph while maintaining an overview of the broader structure [12]. This interactivity fosters trust by making the system's reasoning transparent, as users can see how answers are derived from the graph [13]. Furthermore, visualization supports both open-ended and goal-oriented exploration, helping users generate insights and make decisions by revealing patterns and relationships that might otherwise remain hidden. For instance, participants in a study praised

Wikipedia-style interfaces for on-the-fly entity hopping, highlighting visualization's role in organic discovery [13]. The ability to integrate diverse data sources through visualization is also critical, enabling cross-domain analysis and facilitating knowledge discovery by highlighting relationships, clusters, and outliers [14]. This is particularly important in domains like digital heritage, where visualization interfaces support users in browsing related items and synthesizing narratives [15]. Effective visualization enhances communication for diverse audiences, simplifying potentially compound concepts and fostering collaboration, which is vital for building trust in KG systems [14].

2.3 Prompting Strategies for Knowledge Graph QA

Large language models (LLMs) like GPT-4 and LLaMA-2 can enhance knowledge graph QA (KGQA) by parsing and responding to user queries in natural language. A pivotal question involves how to direct these models to draw upon the KG consistently. Two main strategies—*hard prompting* and *soft prompting*—shape how the model interacts with stored knowledge.

Hard Prompting.

Hard prompting uses explicit textual instructions placed before or together with the user query. For KGQA, such prompts typically specify that the model should only reference known edges in the graph, highlight each logical hop, or refrain from inventing unsupported facts. A sample prompt might read:

> *"You have access to the following knowledge graph. Retrieve only the facts that appear in the graph. Step by step, find the path from 'X' to 'Y' and then provide your final answer."*

By scripting instructions in plain text, hard prompts remain flexible and easy to modify. Few-shot examples can be inserted to illustrate the precise style of reasoning desired. However, reliance on textual instructions alone can lead to inconsistencies when the KG is multiplex, contains self-loops, or the domain requires nuanced knowledge. Minor prompt variations may also induce fluctuating results [16].

Soft Prompting.

Soft prompting trains or fine-tunes a set of "virtual tokens" (embedding vectors) that precede the model's usual input sequence. Rather than providing explicit textual cues, these learned

tokens implicitly guide the model to reference valid graph edges and produce consistent multi-hop reasoning. Early research shows that soft prompting can enhance performance in specialized tasks by encoding domain-specific constraints or logical templates [17, 18]. Despite its potential to yield more stable KG-focused reasoning, soft prompting requires additional training data or curated examples and can be less transparent, since these embeddings are not directly interpretable text.

Comparative Trade-Offs.

While hard prompting excels at rapid prototyping and interpretability (any user can read or modify the textual prompt), it may be too rigid or verbose for specialized domains. Soft prompting, conversely, can internalize domain logic without burdensome prompt engineering, but it introduces training overhead and can lack straightforward human interpretability. In practice, hybrid solutions often combine a concise textual prompt with a small set of specialized embedding tokens, leveraging the strengths of both methods. For KGQA systems focusing on general user queries and minimal domain fine-tuning, hard prompting remains a practical choice, while specialized or mission-critical environments may favor soft prompting's reliability.

2.4 Data Visualization and Interactive Interfaces

Visualization is pivotal for knowledge graph (KG) systems, enabling users to interpret and validate intricate relational data through intuitive representations. By employing visual cues, clustering structures, and interactive exploration, these tools elucidate the topology of nodes and edges, rendering KGs accessible to both domain experts and non-expert users. Effective visualizations must be carefully tailored to specific domains and user needs, balancing clarity with the capacity to reveal hidden patterns, while incorporating interactive features that enhance engagement and foster a deep understanding of the graph's structure and semantics.

Knowledge graphs often utilize high-dimensional node embeddings, derived from techniques like graph neural networks or matrix factorization, to capture semantic relationships. Visualizing these embeddings requires dimensionality reduction to project them into two or three dimensions for human interpretation [19, 20]. Uniform Manifold Approximation and Projection (UMAP) is a robust method that preserves both local and global structures by modeling data as a fuzzy simplicial set. In high-dimensional space, UMAP constructs edge weights between points $\mathbf{x}_i$ and $\mathbf{x}_j$ using a Gaussian kernel adjusted for local density:

$$\rho_{ij} = \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right),$$

where ρ_i is the distance to the nearest neighbor, and σ_i is a scaling factor. In low-dimensional space, weights are defined as:

$$\mu_{ij} = \left(1 + a \cdot d(\mathbf{y}_i, \mathbf{y}_j)^{2b}\right)^{-1},$$

with a and b as fitted hyperparameters. UMAP optimizes the cross-entropy loss:

$$\mathcal{L}_{\text{UMAP}} = \sum_{i,j} \left[\rho_{ij} \log \frac{\rho_{ij}}{\mu_{ij}} + (1 - \rho_{ij}) \log \frac{1 - \rho_{ij}}{1 - \mu_{ij}}\right],$$

yielding clusters that reflect semantic neighborhoods, ideal for KGs with non-linear relationships. t-Distributed Stochastic Neighbor Embedding (t-SNE) emphasizes local structure by modeling pairwise similarities as conditional probabilities. In high-dimensional space, the similarity between points $\mathbf{x}_i$ and $\mathbf{x}_j$ is:

$$p_{j|i} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2\right)},$$

where σ_i is set by a perplexity parameter. The joint probability is symmetrized: $p_{ij} = (p_{j|i} + p_{i|j})/2N$. In low-dimensional space, similarities use a Student's t-distribution:

$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}}.$$

t-SNE minimizes the Kullback-Leibler divergence:

$$\mathcal{L}_{\text{t-SNE}} = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}},$$

highlighting local clusters but often distorting global structure unless carefully tuned. Principal Component Analysis (PCA) is a linear method that projects data onto the eigenvectors of the covariance matrix corresponding to the largest eigenvalues, maximizing variance. For a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, PCA computes the covariance matrix $\mathbf{C} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$, performs eigendecomposition, and projects $\mathbf{X}$ onto the top k eigenvectors to obtain low-dimensional representations $\mathbf{Y} = \mathbf{X}\mathbf{W}$, where $\mathbf{W} \in \mathbb{R}^{d \times k}$ contains the eigenvectors. PCA is computationally efficient but captures only linear relationships, limiting its effectiveness Clustering algorithms organize KG nodes into groups reflecting shared semantics, roles, or relationships, enhancing visualization interpretability. Density-based methods like DBSCAN and HDBSCAN identify clusters as regions of high density in the embedding space, adapting to varying cluster sizes

Table 2.1: Comparison of Dimensionality Reduction Techniques for KG Visualization

Technique	**Strengths**	**Best Use Cases**
UMAP	Balances local/global structure; coherent clusters	Non-linear KGs; general-purpose visualization
t-SNE	Emphasizes local neighborhoods; fine cluster detail	Small KGs; local structure analysis; needs tuning
PCA	Linear; computationally efficient; interpretable	Linear relationships; large datasets; quick prototyping

without requiring a predefined number of clusters [21]. HDBSCAN builds a hierarchical cluster tree by adjusting density thresholds, enabling multi-scale analysis suitable for KGs with layered ontologies. Hierarchical clustering constructs a dendrogram by iteratively merging or splitting nodes based on similarity, aligning with the hierarchical structures common in KGs. Automated labeling of clusters, using the most frequent entity type or relation (e.g., labeling a cluster dominated by *published_in* relations as "Academic"), provides immediate context, guiding users toward relevant semantic domains.

Force-directed graph drawing (FDGD) is a cornerstone of KG visualization, treating edges as springs and nodes as repelling charges to optimize layout [22]. The Fruchterman–Reingold algorithm minimizes an energy function over a graph $G = (V, E)$, balancing attractive forces $f_a(d) = d^2/k$ between connected nodes and repulsive forces $f_r(d) = -k^2/d$ among all nodes, where d is the distance and k is a constant [23]. This produces layouts with uniform edge lengths and clear component separation. Variants like Kamada–Kawai prioritize edge length optimization, yielding different visual emphases. Combining FDGD with an embedding-based initialization (e.g., UMAP coordinates) enhances layouts by aligning them with semantic structures. However, dense KGs risk producing "hairballs"—cluttered displays with overlapping elements—requiring scalable techniques like focus+context views, as in Munzner's H3 system, which emphasizes a focal area while retaining contextual breadth [24].

Interactive tooling transforms static KG visualizations into dynamic exploratory platforms. Frameworks such as `NetworkX` for graph processing, D3.js for web-based visualizations, `PyVis` for Python interactivity, Cytoscape for biological networks, Gephi for network analysis, and KeyLines for enterprise solutions enable real-time layout adjustments [25–28]. Users can filter edges by type or relevance, hover over nodes to access metadata (e.g., entity attributes or relation details), and highlight query-relevant subgraphs. In KGQA, these features illuminate reasoning paths, such as tracing a path from *Newton* to *gravity* to *Principia*, enhancing transparency. Visual enhancements like node grouping by cluster, color-coding by category, or glowing paths emphasize key relationships, enabling users to explore domains like scientific discoveries or cultural networks iteratively.

Effective KG visualizations align with domain-specific needs while balancing clarity and discoverability. Tools like Histropedia or EntiTree offer pre-configured views, such as historical timelines or genealogical trees, simplifying complex data for non-experts [29]. Features like edge bundling to reduce clutter, view switching between node-link and matrix layouts, and context preservation during navigation ensure intuitive exploration [30]. Interactivity, including zooming into dense regions, panning across the graph, and expanding nodes to reveal subgraphs, empowers user-driven discovery [31]. Visualization types impact usability: node-link diagrams suit multi-way, multiplex relationships, matrix visualizations excel for hierarchical data, and treemaps or sunburst charts represent hierarchies effectively. Spatial arrangements, driven by FDGD, create interpretable layouts, with research advocating minimal cognitive load through encodings like spatial position for structure, size for quantities, and color for categories [24]. These principles ensure KG visualizations are both accessible and insightful, fostering trust and engagement across diverse user groups.

2.5 Community Detection in Knowledge Graphs

Community detection algorithms are indispensable for elucidating the structural organization of knowledge graphs (KGs), facilitating efficient navigation, query processing, and visualization. By identifying cohesive sub-networks based on topological properties, these algorithms reveal high-level structures that enhance applications ranging from question-answering (QA) systems to analytical tasks. This section provides a mathematically rigorous exploration of four prominent community detection algorithms—Girvan–Newman, Louvain Method, Infomap, and METIS—detailing their formulations, operational mechanisms, and applications in KGs, while addressing their strengths and limitations.

2.5.1 Girvan–Newman Algorithm

The Girvan–Newman algorithm [32] detects communities by iteratively removing edges with the highest betweenness centrality, effectively severing connections between distinct sub-networks. Betweenness centrality measures an edge's role as a bridge by calculating the fraction of shortest paths passing through it.

The betweenness centrality $C_B(e)$ of an edge e in a graph $G = (V, E)$ is given by:

$$C_B(e) = \sum_{s \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}},$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(e)$ is the

number of those paths traversing edge e.

The algorithm proceeds as follows:

1. Compute $C_B(e)$ for all edges $e \in E$.
2. Remove the edge with the highest betweenness centrality.
3. Recalculate betweenness centralities for the remaining edges.
4. Repeat until no edges remain or the desired number of communities is achieved.

This process generates a dendrogram, revealing the hierarchical community structure. In KGs, Girvan–Newman excels at identifying bridging edges that connect disparate domains, such as interdisciplinary links in a scientific literature KG. However, its computational complexity, $O(m^2n)$ for n vertices and m edges, makes it impractical for large-scale KGs, limiting its use to medium-sized graphs.

2.5.2 Louvain Method

The Louvain Method [33] is a modularity-based algorithm known for its speed and scalability, making it a preferred choice for industrial-scale KGs. It optimizes modularity, a metric that evaluates the density of intra-community edges relative to inter-community edges.

The modularity Q of a partition is defined as:

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j),$$

where A_{ij} is the adjacency matrix entry for nodes i and j, k_i is the degree of node i, m is the total number of edges, c_i is the community assignment of node i, and $\delta(c_i, c_j) = 1$ if $c_i = c_j$, and 0 otherwise.

The algorithm operates in two iterative phases:

1. **Modularization**: Each node starts in its own community. For each node, the algorithm evaluates the modularity gain of moving it to a neighbor's community, adopting the move that maximizes Q. This continues until no further improvement is possible.
2. **Aggregation**: Communities are collapsed into super-nodes, forming a new graph where edges represent inter-community connections. The process repeats on this coarsened graph.

The Louvain Method's efficiency, capable of processing graphs with millions of nodes, stems from its greedy optimization strategy. In KGs, it effectively identifies broad thematic regions, such as product categories in e-commerce graphs. However, its resolution limit may prevent the detection of small, specialized communities in the presence of large clusters, a critical consideration for KGs with diverse scales.

2.5.3 Infomap

Infomap [34] models community detection as a compression problem, using random walk flows to identify communities where walkers remain for extended periods. This approach is particularly suited for KGs with nested or hierarchical structures.

The map equation, which Infomap minimizes, is:

$$L(M) = qH(Q) + \sum_{i=1}^{m} p_i H(P_i),$$

where M is the partition into m communities, q is the probability of a random walker exiting a module, $H(Q)$ is the Shannon entropy of the module visit distribution, p_i is the probability of visiting nodes within module i, and $H(P_i)$ is the entropy of the visit distribution within module i.

Infomap seeks the partition M that minimizes $L(M)$, compressing the description of random walk trajectories. This method excels at uncovering multi-level hierarchies, making it ideal for KGs with nested domains, such as subfields within a research KG. However, its computational overhead, higher than Louvain's, may pose challenges for extremely large graphs.

2.5.4 METIS

METIS [35] is a multilevel partitioning algorithm that achieves high-quality community detection through recursive coarsening, partitioning, and refinement. Its efficiency and scalability make it suitable for large KGs.

The algorithm operates in three phases:

1. **Coarsening**: The graph is recursively contracted by matching nodes to form a smaller graph.

2. **Initial Partitioning**: The coarsest graph is partitioned into the desired number of communities using a bisection algorithm.

3. **Refinement**: The partition is refined during uncoarsening to minimize the cut size, defined as:

$$C = \sum_{i \neq j} w(e_{ij}),$$

where $w(e_{ij})$ is the weight of edge e_{ij} between communities i and j.

METIS's efficiency enables it to handle large KGs, and its integration into frameworks like Cytoscape's CDAPS [36] supports multiscale community detection, enhancing visualization coherence. In KGs, METIS groups semantically related entities, reducing complexity for scalable interfaces.

Applications to KG Exploration.

Community detection reveals broad thematic regions—such as specialized research domains or product categories—within a knowledge graph. Users may leverage these communities to filter queries, focusing on an area of interest or ignoring irrelevant subgraphs. Communities also guide visual mapping, grouping related nodes under shared annotations. In QA scenarios, localizing a query to a specific community can reduce ambiguity, as the system knows to confine its lookups to nodes within that sub-network. As a result, community detection supports targeted exploration, speeds up repeated queries, and uncovers latent structures that might otherwise remain invisible.

2.5.5 Limitations

The following table summarizes the key characteristics of the discussed algorithms:

Table 2.2: Comparison of Community Detection Algorithms for KGs

Algorithm	**Strengths**	**Limitations**	**Suitability for KGs**
Girvan–Newman	Hierarchical, interpretable	High complexity ($O(m^2n)$)	Medium-sized KGs
Louvain	Fast, scalable	Resolution limit	Large-scale KGs
Infomap	Multi-level hierarchies	Higher overhead	Hierarchical KGs
METIS	Efficient, high-quality cuts	Less focus on hierarchies	Large, visualization-driven KGs

The choice of algorithm depends on the KG's size, the need for hierarchical detection, and computational resources. Girvan–Newman's interpretability is offset by its computational cost, while Louvain's speed is tempered by its resolution limit. Infomap's hierarchical detection is ideal for nested structures but requires more resources. METIS offers a balance of efficiency and quality, particularly for visualization tasks.

Recent advancements, such as the Revised Medoid-Shift (RMS) method [37], combine traditional approaches with K-nearest neighbors to address limitations like overlapping communities and scalability. Additionally, integrating node attributes or semantic information, as seen in methods like MRFasGCN [38], can enhance detection accuracy in KGs with rich metadata.

2.6 Hallucination Detection

Large language models have shown versatility in generating plausible-sounding text, but this flexibility can lead to *hallucination*, where a model fabricates or distorts facts. When knowledge graph question answering harnesses LLMs, ensuring factual alignment becomes paramount. Hallucinated answers pose risks to both user trust and subsequent analytical workflows.

Verification Against Ground Truth

A direct strategy for detecting hallucinations is comparing LLM-generated outputs to a trusted reference or the KG's validated subset. If the model's purported answer references entities or relationships that contradict existing graph facts, the system flags it [39]. Even partial ground truth checks—for instance, confirming that the named entities exist in the KG or that the model's relation types match known predicates—can expose inaccuracies. Metrics such as precision, recall, and F_1-scores quantify the alignment between generated statements and correct facts. In QA, *precision at top-k* measures how many high-confidence statements match the KG. Specialized metrics like *ontology consistency* track compliance with domain constraints, providing deeper insight into semantic alignment.

Uncertainty Estimation and Confidence Scoring

Confidence or uncertainty estimation further mitigates hallucination risk. An LLM can be prompted to report a self-assessed confidence level, or an ensemble of prompts may vote on the correctness of each candidate answer. Low-confidence results can be hidden, labeled for review, or retested with alternate phrasing. This mechanism highlights uncertain aspects of the answer or edges in a path, prompting users to scrutinize them carefully [40].

Consistency Checks and Logical Constraints

KGs often encode domain constraints, such as type hierarchies or property restrictions. If an LLM claims an entity is both *Person* and *Chemical*, violating an exclusive domain rule,

the system flags the inconsistency. Similarly, if the chain-of-thought references non-existent edges, it signals potential hallucination. Logical or rule-based checks ensure answers remain within the KG's semantics. Cross-referencing reasoning paths against the KG validates each step, discarding or rechecking paths with discrepancies.

Multi-Pass Reasoning and Iterative Refinement

Prompting the model multiple times or under different constraints catches inconsistencies. If answers vary significantly without KG changes or fail to confirm claimed edges, the system marks contested content for scrutiny [41]. This iterative approach mirrors testing protocols, enhancing stability and reliability.

Impact on Downstream Applications

Hallucinations can contaminate analyses, such as distorting community detection by fabricating links between subgraphs, or misleading experts in critical domains (e.g., medical queries). Integrating real-time hallucination detection ensures KGQA remains trustworthy, preventing cascading inaccuracies.

2.7 Evaluation and Benchmarking for KGQA

Benchmarks like LC-QuAD and WebQuestions [42, 43] evaluate KGQA systems on accuracy, reasoning depth, and fact alignment. They challenge models to map natural language to KG facts, measuring correctness, coverage, and ambiguity resolution. Advanced benchmarks include multi-hop queries, testing systematic traversal and integration of multiple relations. Metrics also emphasize explainability, requiring systems to provide rationales or subgraphs supporting answers. These benchmarks guide development, highlighting strengths and weaknesses in prompting, reasoning, and interface design.

2.8 Conclusion of Related Work

Modern KG research integrates query interfaces, visualization, LLM-driven QA, and hallucination mitigation. Systems like **KG4Vis** and **LinkQ** offer visualization and query refinement but lack fully transparent reasoning displays. Hard prompting ensures interpretable LLM guidance, while soft prompting provides stability for specialized tasks. Visualization techniques—dimensionality reduction, force-directed layouts, interactive tooling—clarify topologies, with tools like Cytoscape and D3.js enabling dynamic exploration. Community detection

(Girvan–Newman, Louvain, Infomap, METIS) reveals thematic regions, enhancing query focus and visualization coherence. Hallucination detection via ground truth verification, uncertainty estimation, and logical checks ensures factual alignment. Benchmarks emphasize accuracy, explainability, and multi-hop reasoning, guiding future innovations. These elements converge to create reliable, user-centric KG systems prioritizing transparency and correctness.

Chapter 3

Methods

TRACE is a **R**easoning and **A**nswer-path **C**omprehension **E**ngine. This chapter delineates the methodologies employed in the development of TRACE, an interactive visualization tool crafted to enhance transparency and accessibility in knowledge graph question answering (KGQA) systems. The focus is exclusively on the visualization and preprocessing components, which empower non-expert users to explore knowledge graphs (KGs) and comprehend reasoning paths. The actual query processing and reasoning path determination, including the use of retrievers such as SubgraphRAG and reinforcement learning-based approaches, are managed by a separate component developed in another thesis. Consequently, this chapter concentrates on the dataset preprocessing, meta-node visualization, query visualization integration, and frontend implementation, which collectively form the backbone of TRACE's user interface. Each section provides an exhaustive account of the techniques, rationales, and implementation details to ensure a comprehensive understanding suitable for a thesis.

3.1 Design Choices

Visualizing large-scale knowledge graphs presents formidable challenges due to their inherent complexity and scale. Traditional visualization techniques, such as force-directed layouts, frequently produce cluttered and unintelligible displays, often termed "hairballs," where overlapping nodes and edges obscure the graph's structure and relationships. Research indicates that graphs exceeding 50 nodes become visually overwhelming, rendering them ineffective for exploration or analysis [44]. This issue is particularly pronounced for non-expert users unfamiliar with graph theory or the specific KG's structure, as they lack the technical background to navigate such dense representations. The complexity is amplified in KGQA systems, where users must not only explore the graph but also verify the reasoning paths behind answers to ensure trust and transparency. To address these challenges, TRACE adopts

a novel visualization strategy that abstracts the KG into a collection of meta-nodes, each encapsulating a semantically coherent community of entities and relations. This abstraction yields a scalable and interpretable overview, enabling users to grasp the KG's semantic diversity without being inundated by its full complexity. By partitioning the KG into communities and representing each as a meta-node, TRACE facilitates high-level exploration while preserving the ability to delve into specific details through interactive mechanisms such as tooltips and query auto-population. This approach is tailored for non-expert users, as it conceals technical intricacies, such as SPARQL queries and RDF structures, allowing focus on the data's semantic content rather than its underlying representation [1]. Moreover, the meta-node framework supports the visualization of reasoning paths derived from KGQA systems, enabling users to trace answer derivation and verify correctness visually, thereby enhancing trust in the system's outputs. The novelty of this method lies in its integration of community detection, semantic embeddings, and dimensionality reduction to create an interactive, scalable interface. Unlike conventional node-link diagrams, which become unwieldy with large graphs, or subgraph-only views, which lack a comprehensive overview, TRACE balances global context with local detail, enhancing both accessibility and transparency.

The decision to use meta-nodes was informed by a thorough evaluation of alternative visualization strategies, each revealing significant limitations. Direct rendering of the entire KG using force-directed algorithms, as implemented in tools like Cytoscape or Gephi, resulted in hairballs where entities clustered tightly, and typed relations, such as discovered or brother_of, were indistinguishable, rendering exploration or reasoning path interpretation impossible [27, 28]. A query-driven approach, which renders only the reasoning subgraph and its path after a user submits a query, reduced visual clutter but required users to understand valid query types, such as multi-hop queries with specific relations, limiting accessibility for non-experts and providing no overview of the KG's broader structure [45]. Sampling-based methods, including random entity sampling or stratified sampling by relation type or entity degree, failed to preserve the semantic coherence of typed relations. For instance, random sampling often overrepresented high-degree hubs, such as United States, while omitting critical entities, such as Newton's Laws, disrupting the KG's relational integrity. The meta-node approach overcomes these shortcomings by abstracting the KG into semantically meaningful clusters, enabling both high-level exploration and detailed interaction, making it an optimal solution for non-expert users in a KGQA context. This strategy aligns with the need for intuitive interfaces that support both open-ended discovery and targeted query validation, ensuring TRACE meets the diverse needs of its users.

3.2 Dataset Preprocessing

The WebQSP knowledge graph, denoted as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, where $\mathcal{E}$ represents the set of entities, $\mathcal{R}$ the set of relations, and $\mathcal{T} = \{(s, r, o) \mid s, o \in \mathcal{E}, r \in \mathcal{R}\}$ the set of triples, serves as the foundational dataset for TRACE's visualization tool [46]. Given the graph's size and density, encompassing thousands of entities and potentially millions of triples, direct visualization is impractical due to computational complexity and perceptual overload. The preprocessing pipeline transforms $\mathcal{G}$ into a manageable, interpretable form by partitioning it into communities, generating textual summaries, embedding communities semantically, and projecting them into a two-dimensional space for visualization, ensuring scalability and user accessibility.

The preprocessing pipeline begins with partitioning $\mathcal{G}$ into $k = 500$ communities using the METIS algorithm, implemented via NetworkX, denoted $\mathcal{C} = \{C_1, C_2, \ldots, C_{500}\}$, where each $C_i \subseteq \mathcal{E}$, $\bigcup_{i=1}^{500} C_i = \mathcal{E}$, and $C_i \cap C_j = \emptyset$ for $i \neq j$ [25, 35]. METIS, a multilevel partitioning algorithm, operates by recursively coarsening the graph to reduce its size, partitioning the coarsened graph using techniques like spectral bisection, and refining the partition through iterative optimization to minimize the edge cut, defined as:

$$\sum_{i \neq j} |\{(s, r, o) \in \mathcal{T} \mid s \in C_i, o \in C_j\}|.$$

This minimization ensures that communities are semantically coherent, prioritizing connections within communities over those between them, which preserves the integrity of typed relations like related_to or author_of. Formally, METIS solves:

$$\min_{\mathcal{C}} \sum_{i \neq j} |\{(s, r, o) \in \mathcal{T} \mid s \in C_i, o \in C_j\}|,$$

subject to balancing constraints $|C_i| \approx |\mathcal{E}|/k$, ensuring communities are of roughly equal size to avoid skewed visualizations. The choice of $k = 500$ was determined through empirical evaluation, testing values from 100 to 1000 to balance granularity and visual clarity. A smaller k (e.g., 100) produced overly coarse communities, merging unrelated entities, while a larger k (e.g., 1000) created fragmented clusters that cluttered the visualization. The value $k = 500$ ensures that the visualization remains interpretable without sacrificing semantic detail, allowing users to discern distinct thematic regions. Due to the scale of the WebQSP dataset and METIS's balancing strategy, each community contains an average of 2134 entities and 24774 triples. This distribution is a consequence of the dataset's size and the algorithm's design, not a deliberate target, but it effectively supports visualization by creating manageable

community sizes. METIS was selected over alternatives like spectral clustering or the Louvain method due to its computational efficiency and superior preservation of local structure, critical for maintaining the semantic coherence of KG relations [47].

Each community C_i is processed to create a meta-node representation suitable for visualization and interaction. The triple set for C_i is defined as $T_i = \{(s, r, o) \in \mathcal{T} \mid s, o \in C_i\}$, capturing all relations within the community. A community string s_i is constructed by concatenating all triples in T_i, ordered by the ascending entity ID of the subject to ensure consistency. Formally, for $T_i = \{(s_{i1}, r_{i1}, o_{i1}), \ldots, (s_{im}, r_{im}, o_{im})\}$, the string is $s_i = \text{concat}((s_{i1}, r_{i1}, o_{i1}), \ldots, (s_{im}, r_{im}, o_{im}))$. This textual representation encapsulates the community's semantic content, serving as a unified input for subsequent processing steps.

The community string s_i undergoes two critical processing steps to enable visualization and user interaction. First, the DeepSeek R1 large language model generates a concise title t_i (five words or fewer) and description d_i (fifteen words or fewer) to summarize the community's topic, using the prompt shown in Figure 3.1. This prompt is carefully designed to elicit succinct, user-friendly summaries, such as a title "Classical Music" and description "Composers, symphonies, and performances" for a community centered on musical relations. To mitigate potential failures in LLM responses, calls are wrapped in try-except clauses and retried up to eight times, ensuring a success probability of $1 - (1 - 0.92)^8 \approx 1 - 10^{-9}$ [48]. This robust approach guarantees reliable summary generation, which is essential for providing users with intuitive, human-readable context about each community. Second, Sentence-BERT, implemented via PyTorch, embeds s_i into a 384-dimensional vector $\mathbf{e}_i \in \mathbb{R}^{384}$, capturing the community's semantic essence [49, 50]. Sentence-BERT leverages a siamese BERT architecture trained on sentence similarity tasks, producing embeddings that preserve semantic relationships, enabling accurate categorization and visualization of community similarities.

```
Given the knowledge graph triples: [TRIPLES], generate a title
(5 words or fewer) and description (15 words or fewer)
summarizing the community's topic. Answer only with a JSON
object that has a 'title' and 'desc' field.
```

Figure 3.1: DeepSeek R1 prompt for generating a community title t_i and description d_i, where `[TRIPLES]` is the community string s_i.

The community embedding $\mathbf{e}_i$ serves dual purposes: categorization and visualization. For categorization, each community is assigned to one of ten predefined categories aligned with Freebase's topic hierarchy to ensure user-interpretable mappings [51]:

$$\mathcal{K} = \left\{ \begin{array}{l} \text{Arts, Technology, Literature, Science,} \\ \text{Sports, Business, History, Geography, Politics, Entertainment} \end{array} \right\}.$$

Category embeddings $\mathbf{e}_k \in \mathbb{R}^{384}$ are computed using Sentence-BERT, and C_i is assigned to the category $k^* = \arg\max_{k \in \mathcal{K}} \cos(\mathbf{e}_i, \mathbf{e}_k)$, where cosine similarity is defined as:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}.$$

This categorization enables intuitive color-coding and labeling in the visualization, allowing users to quickly identify thematic clusters, such as a group of nodes categorized as Literature versus Science. For visualization, the embeddings are projected into a two-dimensional space using Uniform Manifold Approximation and Projection (UMAP), implemented via the `umap-learn` library [19]. UMAP is a dimensionality reduction technique that preserves both local and global structures, making it ideal for visualizing semantic relationships in KGs. It constructs a high-dimensional graph where nodes represent community embeddings, and edges reflect similarity measured via a Gaussian kernel. This graph is represented as a fuzzy simplicial set, capturing topological structure. A low-dimensional graph is then optimized to approximate this structure by minimizing the cross-entropy between the high-dimensional and low-dimensional simplicial sets. Formally, for high-dimensional data points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and low-dimensional representations $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$, the edge weight between $\mathbf{x}_i$ and $\mathbf{x}_j$ is:

$$\rho_{ij} = \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right),$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance, ρ_i is the distance to the nearest neighbor, and σ_i is a scale parameter ensuring local connectivity. For low-dimensional points, the weight is:

$$\mu_{ij} = \left(1 + a \cdot d(\mathbf{y}_i, \mathbf{y}_j)^{2b}\right)^{-1},$$

where a and b are hyperparameters fitted to the data. UMAP minimizes the cross-entropy loss:

$$\mathcal{L} = \sum_{i,j} \left[\rho_{ij} \log \frac{\rho_{ij}}{\mu_{ij}} + (1 - \rho_{ij}) \log \frac{1 - \rho_{ij}}{1 - \mu_{ij}}\right].$$

In TRACE, UMAP is configured with hyperparameters $n_{\text{neighbors}} = 15$, min_dist $= 0.1$, and $n_{\text{components}} = 2$, balancing local neighborhood preservation with global structure to produce coherent, visually distinct clusters. Each community C_i is represented as a meta-

node $m_i = (t_i, d_i, k_i, (x_i, y_i))$, where (x_i, y_i) is the 2D coordinate obtained from UMAP, ready for visualization.

3.3 Visualization of Meta-Nodes

The meta-node visualization forms the primary interface for exploring the KG, providing a high-level overview that enables users to navigate its semantic landscape intuitively. This interface is rendered on a canvas using D3.js, a JavaScript library for creating data-driven documents, which offers precise control over the placement and styling of visual elements [26]. Each meta-node m_i is displayed as a circle positioned at coordinates (x_i, y_i), with its color determined by its category k_i. Ten distinct colors, such as blue for Science, red for Arts, and green for Sports, ensure visual distinguishability and align with the categorical structure of the KG. For each category $k \in \mathcal{K}$, the centroid is computed to position a label:

$$(\bar{x}_k, \bar{y}_k) = \left(\frac{1}{|M_k|} \sum_{m_i \in M_k} x_i, \frac{1}{|M_k|} \sum_{m_i \in M_k} y_i \right),$$

where $M_k = \{m_i \mid k_i = k\}$. The category name, such as Literature or Geography, is rendered at the centroid, providing a clear label for each cluster of meta-nodes. This scatterplot layout visually groups related communities, with clusters indicating semantic proximity—for example, a dense cluster of Literature meta-nodes near History reflects their topical overlap, aiding users in identifying related thematic regions.

Interactive features are integral to TRACE's usability, particularly for non-expert users who may lack familiarity with KG structures. When a user hovers over a meta-node m_i, a tooltip appears, displaying the title t_i, description d_i, the number of entities $|C_i|$, the number of triples $|T_i|$, 50 randomly sampled triples from T_i, and a prompt to click/generate a topic-specific question. For instance, a Science community might show triples like (Einstein, developed, Relativity) or (Newton, discovered, Gravity), offering a snapshot of its content that helps users understand the community's scope. The tooltip's prompt encourages the user to click to generate a question, reducing the cognitive barrier to query formulation. Specifically, clicking meta-node m_i samples a triple $\tau_i \in T_i$ and auto-populates the query field with a question q_{τ_i}, such as "What is s related to?" for $\tau_i = (s, \text{related\_to}, o)$. For example, clicking a Literature meta-node might generate "What is Shakespeare related to?", enabling users to explore related entities like plays or contemporaries. This mechanism aids non-expert users in formulating valid queries without prior knowledge of the KG's structure, making TRACE accessible to novices while retaining depth for advanced users. The design draws inspiration from tools like Histropedia and EntiTree, which use pre-configured views to sim-

plify exploration, but extends this by dynamically generating interactive, community-based representations tailored to the WebQSP dataset [29].

The meta-node visualization addresses the limitations of prior visualization approaches by providing a scalable abstraction that avoids hairballs while enabling detailed interaction. Unlike static visualizations, which overwhelm users with raw graph complexity, or query-driven displays, which require predefined questions, TRACE's interface supports both open-ended exploration and guided query formulation. The scatterplot's design ensures that users can quickly identify areas of interest, such as a cluster of Politics nodes, and drill down into specific communities through interactive features, fostering an intuitive and transparent exploration experience that enhances user engagement and trust.

3.4 Query Processing for Visualization

The query processing pipeline in TRACE is responsible for handling user queries, preparing them for visualization, and rendering the resulting subgraphs and reasoning paths, while deferring the actual answer generation and path determination to an external component developed in another thesis. This section details the steps involved in query input, clause splitting, integration with the external component, and visualization rendering, ensuring that TRACE presents the reasoning process to users in a clear and transparent manner.

When a user submits a query q through a text field on the TRACE interface, the system processes it to align with the KG's relational structure. The first step is clause splitting, performed using the DeepSeek R1 large language model with the prompt shown in Figure 3.2. This prompt instructs the model to decompose the natural language query into a sequence of clauses, each corresponding to a typed edge in the KG, represented in the form $(entity1, edge_type, entity2)$. The prompt specifies that the response should be a comma-separated list of edge types, ensuring a structured output suitable for downstream processing. For example, the query "the sister of the brother of the man who discovered gravity" yields clauses `discovered,brother_of,sister_of`, reflecting the sequential relations required to traverse the KG from an entity associated with discovered (e.g., Newton) to brother_of and finally to sister_of. To ensure robustness, LLM calls are wrapped in try-except clauses and retried up to eight times, leveraging the same high success probability as in title generation ($1 - (1 - 0.92)^8 \approx 1 - 10^{-9}$) [48]. This approach guarantees reliable clause extraction, which is critical for accurate query processing and subsequent visualization.

The extracted clauses are forwarded to the external component, which processes them to retrieve a subgraph $\mathcal{G}_q \subseteq \mathcal{G}$ and a highlighted reasoning path P. The subgraph is constrained to exactly 50 entities and their associated triples, a limit chosen to ensure visual clarity and

```
You are a reasoning assistant, designed to break down natural
language queries into structured clauses corresponding to typed edges
and entities in a knowledge graph. Given: Available edge types: {R}.
A user's natural language query: '[QUERY]'. Your task:
1. Break down the query into a sequence of clauses, where each clause
represents exactly one edge from the provided edge types.
2. Each clause should be in the form (entity1, edge_type, entity2),
chaining logically to represent the meaning of the query.
3. For each entity in your clauses, provide a plausible and specific
real-world example of such an entity. Respond ONLY with a comma-separated
sequence of clauses in the form [edge_name1, edge_name2, ...]

Example output:
brother_of,married_to,roommates_with
```

Figure 3.2: DeepSeek R1 prompt for clause splitting, where `{R}` is the set of all relations $\mathcal{R}$ in $\mathcal{G}$.

prevent hairball effects, as subgraphs exceeding 50 nodes become difficult to interpret [44]. The reasoning path P represents the sequence of edges that answers the query, such as a path from Newton to a sibling via discovered and brother_of relations. TRACE receives this output and renders $\mathcal{G}_q$ using `react-force-graph`, a React-based library for force-directed graph visualization that supports dynamic, interactive layouts [52]. The subgraph is centered on the canvas, with edges in P highlighted in green and thickened to distinguish them from other connections, making the reasoning process visually explicit. For example, a query about literary influences might highlight a path from Shakespeare to Hamlet to modern adaptations, with thickened green edges emphasizing the traversal. Users can interact with the visualization by adjusting edge thickness to focus on high-relevance connections, filtering edges based on relevance scores provided by the external component, or modifying the force-directed layout's springiness to optimize node separation and reduce overlap. These controls enable users to tailor the visualization to their needs, balancing detailed inspection of the reasoning path with broader contextual exploration of the subgraph.

The query processing pipeline integrates seamlessly with the meta-node interface, allowing users to transition fluidly from high-level exploration to query-specific visualization. For instance, a user might explore a Literature meta-node, click to generate a query about Jane Austen, and then view a subgraph showing connections to her novels and contemporaries. By consolidating clause splitting, external component integration, and visualization rendering within this section, TRACE ensures that all query-related processes are presented cohesively, clarifying their role in the visualization workflow. This design enhances transparency, as

users can visually trace how their query is processed and answered, fostering trust in the KGQA system and supporting both novice and advanced users in their exploration.

3.5 Frontend Implementation Details

The frontend of TRACE is implemented using CSS and TypeScript, leveraging modern web technologies to deliver a responsive, interactive user interface that supports both novice and advanced users. The meta-node scatterplot, which serves as the primary exploration interface, is rendered using D3.js, a JavaScript library that provides fine-grained control over data-driven visualizations, enabling precise placement and styling of meta-nodes [26]. Subgraph visualizations, displayed in response to user queries, are handled by `react-force-graph`, a React-based library that supports dynamic, force-directed layouts optimized for web applications, ensuring smooth rendering of complex subgraphs [52]. These technologies were chosen for their flexibility and performance, critical for handling the large-scale, interactive visualizations required by TRACE.

Preprocessing, which includes community detection, title and description generation, and embedding computations, is performed offline in a Jupyter notebook using Python libraries to ensure efficiency and reproducibility. NetworkX is used for graph manipulation and METIS partitioning, providing a robust framework for processing the WebQSP knowledge graph [25]. Sentence-BERT embeddings are computed using PyTorch, leveraging its efficient tensor operations to handle the high-dimensional embedding process [50]. UMAP projections are executed via the `umap-learn` library, ensuring high-quality dimensionality reduction that preserves semantic relationships [19]. The DeepSeek R1 model, used for title, description, and clause splitting tasks, is integrated into the preprocessing and query processing pipelines, with robust error handling to guarantee reliability. The preprocessed meta-node data, including titles, descriptions, categories, and 2D coordinates, are stored as JSON files for efficient loading by the frontend, minimizing runtime computation and ensuring a responsive user experience.

The frontend dynamically fetches the preprocessed meta-node data, rendering the scatterplot and handling user interactions in real time. User queries are processed by sending clause data to the external component via an API, which returns the subgraph and reasoning path for visualization. The integration of these components ensures a seamless transition from exploration to query visualization, with interactive features like tooltips, query auto-population, and subgraph controls enhancing usability. The following table summarizes the technologies employed in TRACE, highlighting their roles in the system's architecture:

This implementation ensures that TRACE is both scalable and user-friendly, enabling non-

Table 3.1: Technologies Used in TRACE

Component	**Technology**
Community Detection	METIS (via NetworkX)
Title/Description Generation	DeepSeek R1
Community Embedding	Sentence-BERT (via `torch`)
2D Projection	UMAP (via `umap-learn`)
Meta-Node Visualization	D3.js
Subgraph Visualization	`react-force-graph`
Frontend Framework	CSS/TypeScript

expert users to explore KGs and verify reasoning paths while maintaining the rigor required for advanced applications. The modular design, with clear separation of preprocessing, visualization, and query processing, facilitates future extensions, such as integrating new datasets or visualization techniques, and supports seamless integration with the external reasoning component, ensuring a cohesive KGQA system.

Chapter 4

Evaluation

This chapter evaluates the knowledge graph visualization tool developed in this thesis, assessing its usability, effectiveness, and limitations in supporting query answering through interactive, graph-based representations. The evaluation comprises two core components: evaluation metrics, which establish the assessment criteria, and experiments, which detail empirical methodologies and findings. Experiments include a user study conducted at the Massachusetts Institute of Technology (MIT) to examine user interactions and a pipeline evaluation to measure system performance. Results are reported separately, providing quantitative metrics, statistical analyses, and qualitative insights. The chapter aims to rigorously assess the tool's capabilities, identify areas for improvement, and inform the design of user-centric knowledge graph visualization systems.

4.1 Evaluation Metrics

This section outlines the criteria for evaluating the knowledge graph visualization tool, providing a foundation for the experiments in Section 4.2. The metrics assess three critical dimensions: usability, effectiveness, and system performance. Usability metrics evaluate the tool's ease of use and clarity of visualizations, ensuring accessibility for users with varying expertise. Effectiveness metrics measure the accuracy and coherence of query answers, reflecting the tool's core functionality in knowledge exploration. System performance metrics assess computational efficiency and scalability, essential for deploying the tool on large-scale knowledge graphs. These metrics collectively ensure a comprehensive evaluation, addressing both human-centric and technical aspects.

Table 4.1 lists the metrics, their descriptions, and justifications for their inclusion. Each metric is tailored to specific components of the experiments, including the user study (Section 4.2.2) and pipeline evaluation (Section 4.2.1), providing a structured framework for assessing

the tool's performance.

Table 4.1: Summary of Evaluation Metrics

Category	**Metric**	**Description**	**Justification**
User Study	Ease of Use	User-rated ease of deriving answers (1 = very easy, 5 = very difficult) via the interactive interface.	Assesses usability by measuring how intuitively users can navigate and interpret visualizations, critical for accessibility across technical and non-technical users.
User Study	Usefulness of Path	User-rated usefulness of highlighted reasoning paths (1 = not useful, 5 = very useful) in answering queries.	Evaluates the clarity and relevance of visualized reasoning paths, ensuring the tool effectively communicates query answers to users.
User Study	Answer Accuracy	Percentage of correct answers provided by users for correct and incorrect questions.	Measures effectiveness by assessing the tool's ability to support accurate query answering and error detection, a core objective of the system.
Visualization	Preprocessing Time	Time (seconds) to process knowledge graphs through the preprocessing pipeline, including dimensionality reduction.	Evaluates system performance by measuring computational efficiency, crucial for scalability on large graphs in real-world applications.
Pipeline	Query Processing Time	Time (minutes) to process and visualize query responses, varying by hop count.	Assesses system performance for real-time usability, ensuring timely responses for interactive query answering.
Pipeline	Path Validity	Percentage of reasoning paths with semantically valid relation sequences, verified against WebQSP ontology.	Measures effectiveness by evaluating the coherence of visualized reasoning paths, ensuring logical and meaningful query answers.
Pipeline	Path Correctness	Percentage of valid paths with terminal nodes matching ground-truth answers.	Assesses effectiveness by confirming the accuracy of reasoning paths, a key indicator of the tool's reliability in delivering correct answers.

The user study employs ease of use, usefulness of path, and answer accuracy to evaluate human-centric performance. Ease of use captures the intuitiveness of the interface, particularly for users with varying experience, as tested with MIT's undergraduate and postgraduate

participants. Usefulness of path assesses whether highlighted reasoning paths guide users effectively, a critical factor in query comprehension. Answer accuracy measures the tool's ability to support correct answers and error detection, reflecting its reliability across correct and incorrect scenarios.

The pipeline evaluation uses preprocessing time, query processing time, path validity, and path correctness to assess technical performance. Preprocessing time evaluates the efficiency of graph preparation, vital for handling large datasets like WebQSP. Query processing time measures response speed, ensuring the tool meets real-time interaction needs. Path validity and correctness assess the quality of reasoning paths, ensuring semantic coherence and accuracy, which are central to the tool's effectiveness.

These metrics provide a robust framework for the experiments, enabling a thorough evaluation of TRACE's capabilities and limitations, and guiding improvements for user-centric knowledge graph visualization.

4.2 Experiments

This section describes the experimental methodologies used to evaluate the tool, comprising a user study to assess usability and effectiveness from a human perspective and a pipeline evaluation to analyze computational performance. The user study tests participants' ability to answer queries and detect errors using the interactive interface, while the pipeline evaluation examines system efficiency and accuracy. Together, these experiments offer complementary insights into the tool's performance.

4.2.1 Pipeline

Preprocessing

The preprocessing pipeline, a critical component of TRACE, is evaluated for scalability. As discussed in Section 3.1, Uniform Manifold Approximation and Projection (UMAP) was selected as the primary dimensionality reduction algorithm. For benchmarking, we also evaluated t-distributed Stochastic Neighbor Embedding (t-SNE), despite its higher computational complexity.

The experimental setup tests scalability across synthetic knowledge graphs with varying nodes ($n \in [500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 500000, 1000000]$), edges ($m \in [1000, 5000, 10000, 50000, 500000, 2500000, 10000000]$), and community counts ($k \in [500, 1000]$), as described in Section 3.2. The full Cartesian product of these parameters with

UMAP and t-SNE yields 280 configurations. Each configuration was processed, recording total preprocessing time in seconds.

Experiments ran on a high-performance server with an Intel Xeon 32-core CPU, 128 GB of RAM, and an NVIDIA A100 GPU. The software stack included NetworkX for METIS partitioning, PyTorch for Sentence-BERT embeddings, DeepSeek R1 for summarization, and `umap-learn` and `scikit-learn` for UMAP and t-SNE, respectively. Fixed random seeds ensured reproducibility, and each configuration was executed three times, reporting the average time. Results are detailed in Appendix B.1.

Coherence

The coherence experiments assess TRACE's visualization pipeline using 500 queries from the SubgraphRAG test set [53] on the WebQSP knowledge graph, categorized by complexity: 150 1-hop, 150 2-hop, and 200 3-hop queries. Each query is processed to retrieve a subgraph, compute a reasoning path $P = (e_1, r_1, e_2, \ldots, r_{n-1}, e_n)$, and render it visually, where $e_i \in \mathcal{E}$, $r_i \in \mathcal{R}$, and $(e_i, r_i, e_{i+1}) \in \mathcal{T}$. The path's English readout concatenates relations r_i into a natural language statement (e.g., "entity e_1 is related to e_2 via r_1, which leads to e_3 via r_2").

Two metrics are evaluated. *Validity* checks if the path's English readout forms a semantically valid reasoning argument, verified manually against WebQSP's ontology. A path is valid if its relation sequence $r_1, r_2, \ldots, r_{n-1}$ supports a logical chain. *Correctness* assesses whether a valid path's terminal node e_n matches the ground-truth answer a. Correct paths are a subset of valid paths, requiring both validity and accuracy. Processing time and outcomes were recorded for each query.

4.2.2 User Study

The user study, conducted at MIT, evaluates the usability and effectiveness of the tool in query answering. Twelve participants (5 undergraduates, 7 postgraduate researchers) with technical familiarity with graph-based interfaces were selected, ensuring varied analytical experience to assess accessibility across expertise levels.

Participants answered six randomized questions of three difficulty levels: easy (1-hop), medium (2-hop), and difficult (3-hop). Three questions yielded correct answers, and three were incorrect, testing participants' ability to verify correct paths and identify errors. For each question, participants provided the answer, rated ease (1 = very easy, 5 = very difficult) and usefulness of the highlighted path (1 = not useful, 5 = very useful), and offered qualitative feedback on visualization and interactivity. General questions addressed reasoning path visualization, dataset representation, and features like tooltips and auto-populated questions.

The study occurred on May 7, 2025, from 14:40:30 to 14:54:47, with anonymous responses collected using unique identifiers. The controlled MIT setting ensured consistent access, with randomized question order and balanced correct/incorrect questions providing a robust evaluation framework.

Limitations include the small sample size (12 participants), limiting generalizability, and the technical MIT cohort, which may not reflect non-technical users. Predefined questions may not capture real-world query diversity, and the survey format may miss nuanced usability issues. These are considered in the results to contextualize findings.

4.3 Results

4.3.1 Preprocessing

The preprocessing pipeline's scalability was evaluated across 280 configurations, comparing UMAP and t-SNE. The following figures analyze preprocessing times, focusing on trends with graph size, community count, and algorithm choice, informing TRACE's deployment.

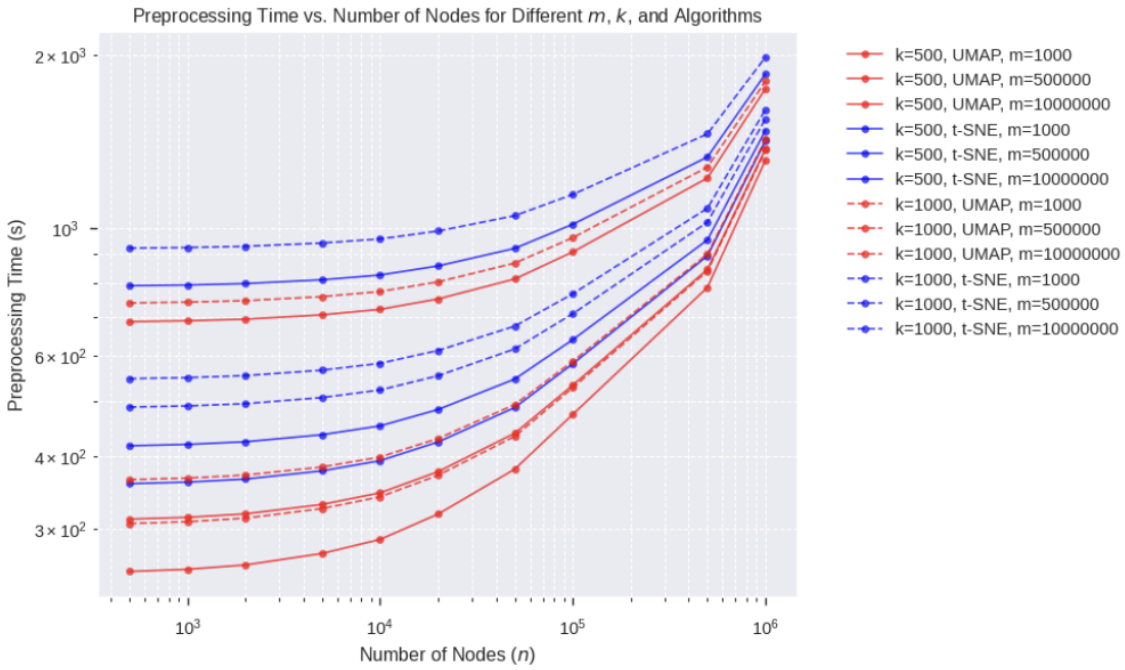


Figure 4.1: Preprocessing time versus number of nodes (n) for selected edge counts (m), with $k \in \{500, 1000\}$, UMAP (red), and t-SNE (blue). Solid lines: $k = 500$; dashed lines: $k = 1000$.

Figure 4.1 shows preprocessing time scaling with nodes (n) for select edge counts m. For $k = 500$, UMAP (in red) rises from 252.34 seconds ($n = 500, m = 1000$) to 1746.89

seconds ($n = 1000000, m = 10000000$), a 6.92-fold increase. t-SNE (in blue) reaches 1853.34 seconds, 6.1% higher. For $k = 1000$, UMAP's time increases to 1800.34 seconds (3.1% rise). The near-linear log-space trend aligns with METIS partitioning ($O(n + m)$) and embedding ($\propto m/k$) costs. UMAP's advantage supports its use in TRACE.

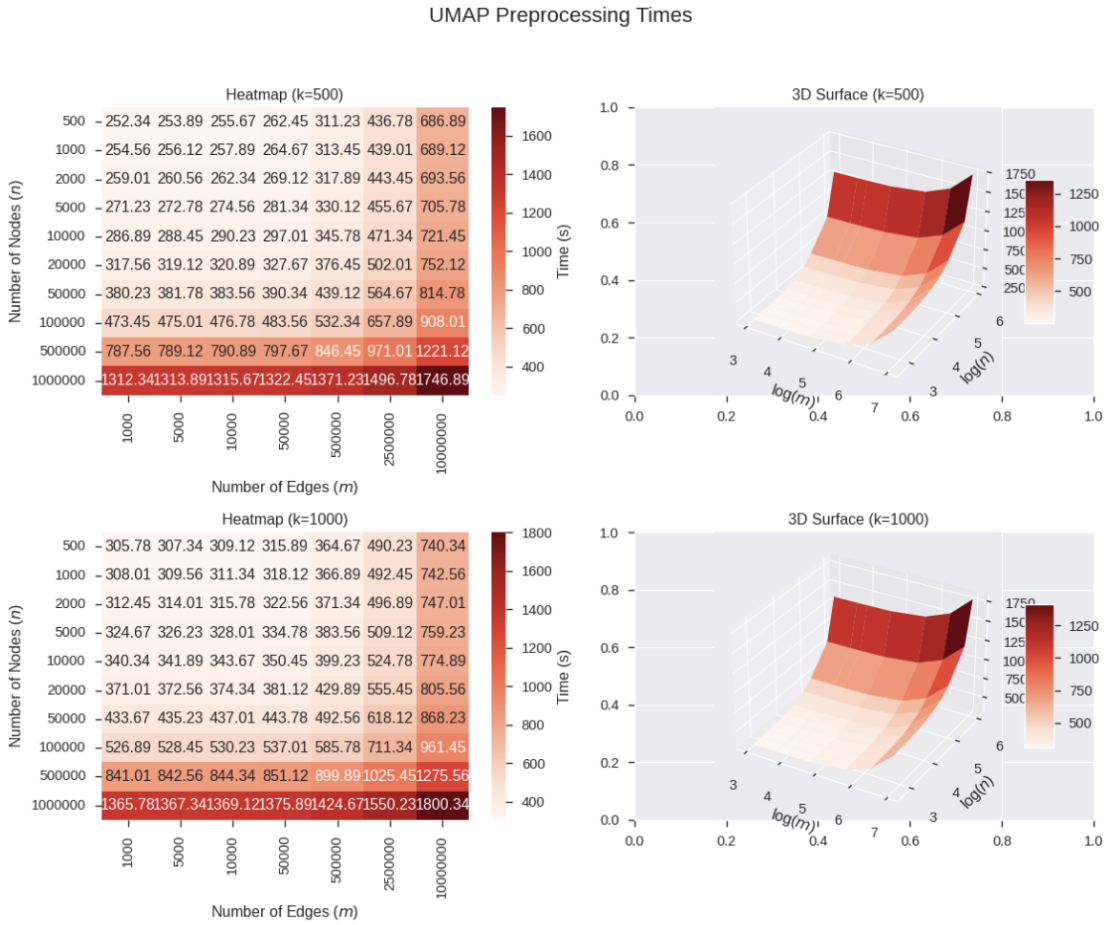


Figure 4.2: UMAP preprocessing times for $k = 500$ (top) and $k = 1000$ (bottom). Left: Heatmaps of time versus n and m. Right: 3D surfaces interpolating time over $\log(n)$ and $\log(m)$.

Figures 4.2 (UMAP) and 4.3 (t-SNE) depict times via heatmaps and 3D surfaces. For UMAP at $k = 500$, time rises from 252.34 seconds ($n = 500, m = 1000$) to 686.89 seconds ($n = 500, m = 10000000$), a 172% increase. At $k = 1000$, time reaches 740.34 seconds (7.8% rise). t-SNE shows higher times: 793.34 seconds at $k = 500$ (15.5% above UMAP) and 922.23 seconds at $k = 1000$ (24.6% above). UMAP's smoother scaling suits TRACE for dense graphs.

Figure 4.4 shows time distributions. For $k = 500$, UMAP peaks at 400 seconds (252.34–1746.89 seconds), t-SNE at 500 seconds (to 1853.34 seconds). At $k = 1000$, UMAP

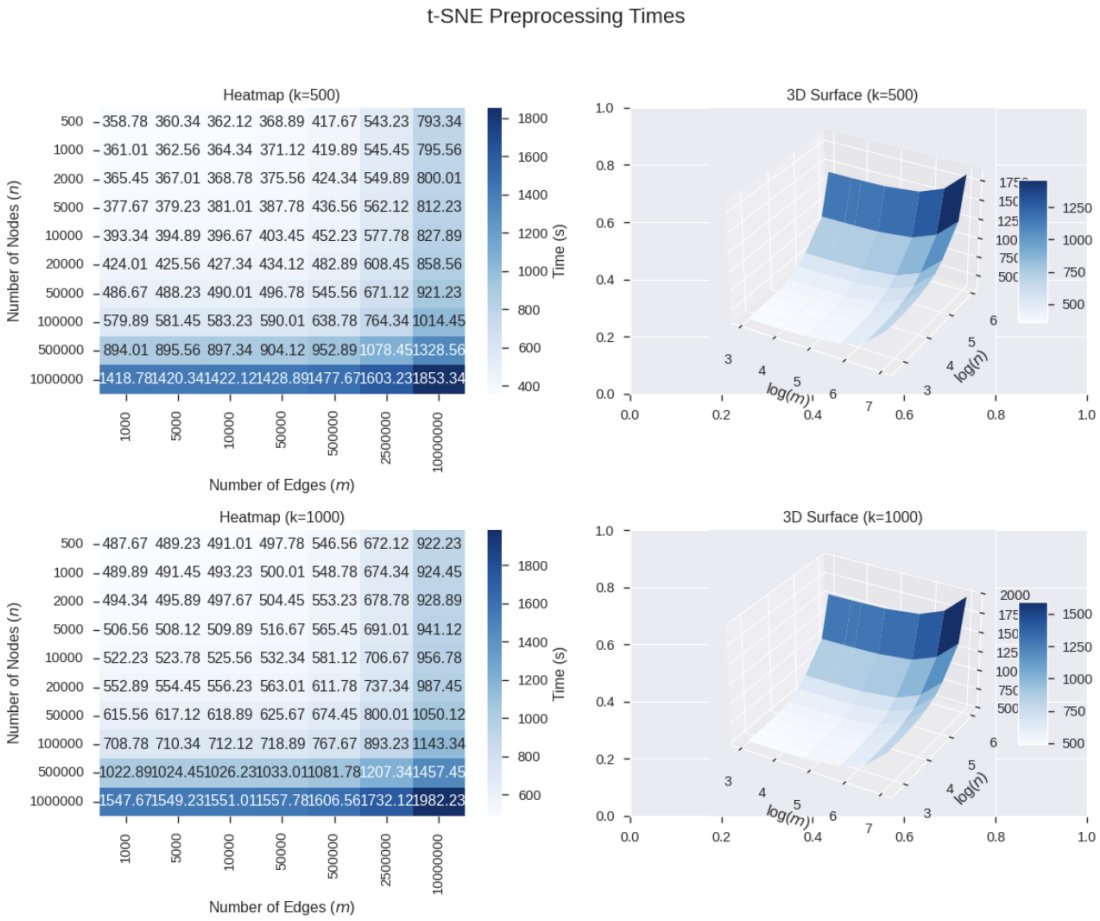


Figure 4.3: t-SNE preprocessing times for $k = 500$ (top) and $k = 1000$ (bottom). Left: Heatmaps of time versus n and m. Right: 3D surfaces interpolating time over $\log(n)$ and $\log(m)$.

peaks at 450 seconds (max 1800.34 seconds), t-SNE at 600 seconds (max 1982.23 seconds). UMAP's narrower distribution ensures stable latency.

Figure 4.5 examines time versus edges (m). For $k = 500$, UMAP's trendline has a 200-second slope per decade, t-SNE's is 250 seconds. At $k = 1000$, slopes are 210 (UMAP) and 280 (t-SNE). UMAP's lower slope supports dense graph handling.

UMAP's lower times and predictable scaling confirm its suitability for TRACE, while t-SNE's quadratic complexity poses challenges for frontend deployment.

4.3.2 Coherence

The coherence experiments evaluate TRACE's visualization pipeline across 500 queries, analyzing processing times, *Validity*, and *Correctness*. Summary statistics are in Table 4.2, with details in Appendix B.2.

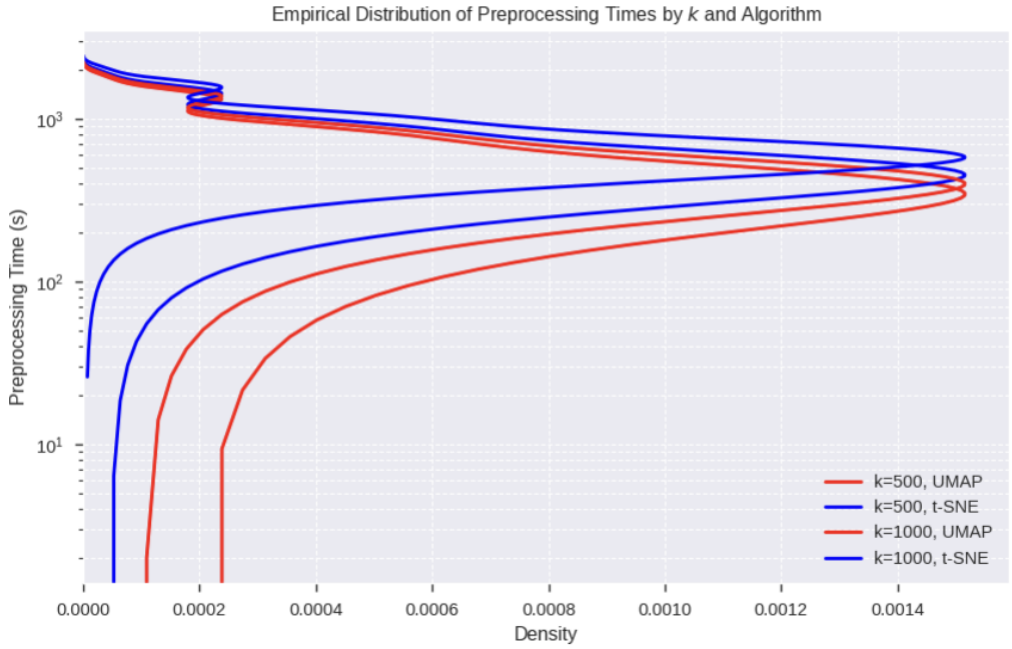


Figure 4.4: Empirical distribution of preprocessing times for $k \in \{500, 1000\}$, UMAP (red), and t-SNE (blue).

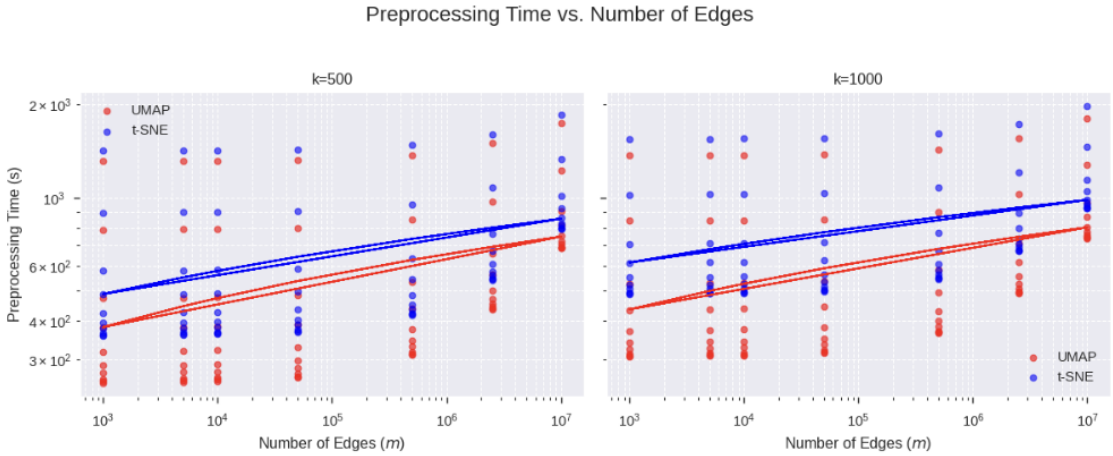


Figure 4.5: Preprocessing time versus number of edges (m) for $k = 500$ (left) and $k = 1000$ (right), UMAP (red) and t-SNE (blue), with logarithmic trendlines.

Table 4.2 shows processing times increasing from 0.54 minutes (1-hop) to 1.70 minutes (3-hop), a 3.15-fold rise. *Validity* drops from 84.0% (126/150, 1-hop) to 70.0% (140/200, 3-hop). *Correctness* peaks at 74.0% (111/150, 2-hop), dipping to 67.5% (135/200, 3-hop).

Figure 4.6 shows 1-hop times centered at 0.54 minutes, 2-hop at 1.12 minutes, and 3-hop at 1.70 minutes, with wider spreads for complex queries. Optimizations like caching could improve real-time performance.

Hop Type	Time (min)	Val. Paths	Val. (%)	Corr. Paths	Corr. (%)
1-Hop (150 queries)	0.54	126	84.0	108	72.0
2-Hop (150 queries)	1.12	120	80.0	111	74.0
3-Hop (200 queries)	1.70	140	70.0	135	67.5

Table 4.2: Query Processing Metrics by Complexity: 1-hop queries fastest + easiest (0.54 min, 84% valid), 3-hop slowest + hardest (1.70 min, 70% valid).

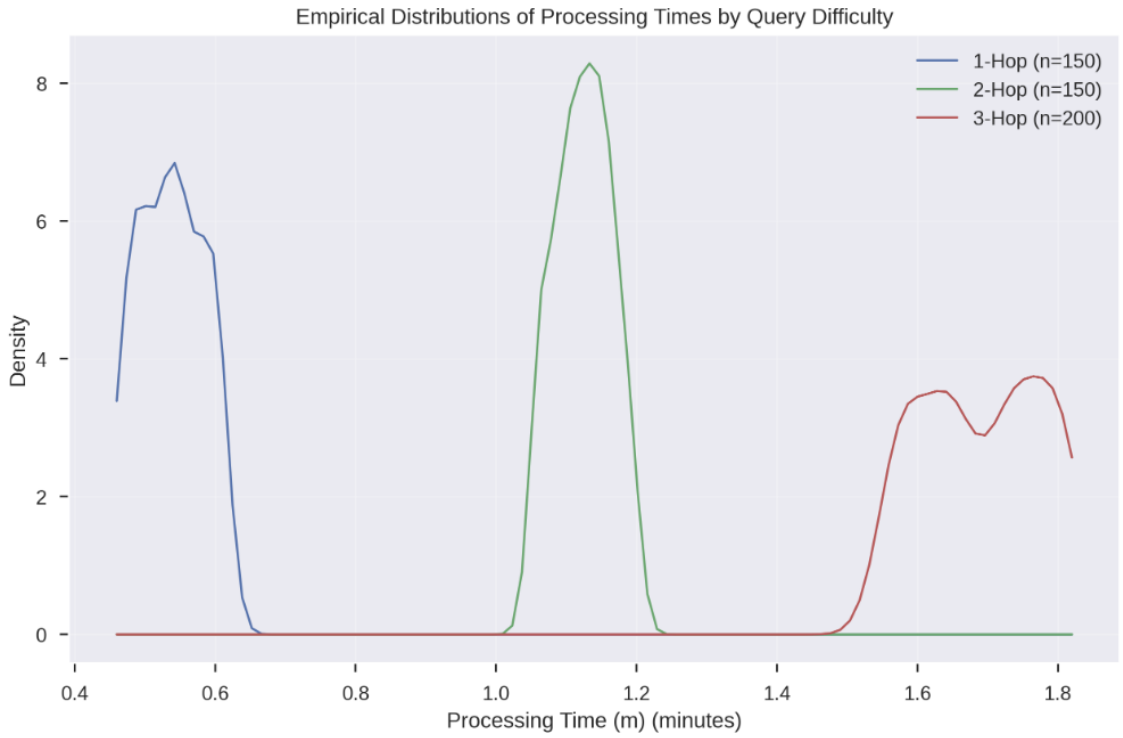


Figure 4.6: Distribution of query processing times (in minutes) by hop count for 1-hop, 2-hop, and 3-hop queries.

Figure 4.7 shows 386 valid paths (77.2%), with 354 correct (70.8%), yielding 91.7% correctness among valid paths. TRACE's high correctness among valid paths indicates effective visualization, but 3-hop query challenges suggest optimization needs.

4.3.3 User Study

The user study assesses usability and effectiveness among 12 MIT participants (5 undergraduates, 7 postgraduates). Quantitative metrics, statistical analyses, and qualitative feedback are stratified by group and question difficulty.

Both groups achieved 100% accuracy for easy and medium correct questions. For difficult correct questions, accuracy was 80% (undergraduates) and 85.7% (postgraduates). For

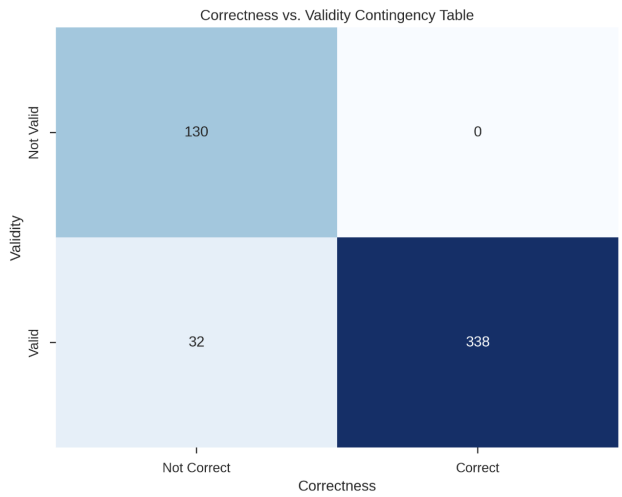


Figure 4.7: Correctness-Validity matrix for all 500 queries, showing the relationship between *Validity* and *Correctness* outcomes.

Diff.	**Corr.**	**Ease (1–5)** ↓	**Usef. (1–5)** ↑	**Acc. (%)**
Easy (1-hop)	Correct	1.20	4.80	100
Easy (1-hop)	Incorrect	1.40	4.60	80
Medium (2-hop)	Correct	1.80	4.20	100
Medium (2-hop)	Incorrect	2.00	4.00	60
Difficult (3-hop)	Correct	2.60	3.60	80
Difficult (3-hop)	Incorrect	3.00	2.80	40

Table 4.3: Undergraduate Results by Difficulty: Simple queries easy (1.20–1.40 ease, 4.60–4.80 usefulness, up to 100% accuracy), difficult queries less accurate (40–80%, 2.80–3.00 ease).

Diff.	**Corr.**	**Ease (1–5)** ↓	**Usef. (1–5)** ↑	**Acc. (%)**
Easy (1-hop)	Correct	1.14	4.71	100
Easy (1-hop)	Incorrect	1.29	4.57	85.7
Medium (2-hop)	Correct	1.57	4.29	100
Medium (2-hop)	Incorrect	1.86	4.14	71.4
Difficult (3-hop)	Correct	2.29	3.86	85.7
Difficult (3-hop)	Incorrect	2.71	3.14	57.1

Table 4.4: Postgraduate Results by Difficulty: Simple queries still easy (1.14–1.29 ease, 4.57–4.71 usefulness, up to 100% accuracy), difficult queries less accurate (57.1–85.7%, 2.29–2.71 ease). Graduates found the tool less useful than undergrads.

incorrect questions, accuracy was 80% and 85.7% for easy, 60% and 71.4% for medium, and 40% and 57.1% for difficult. Postgraduates outperformed undergraduates.

Ease ratings increased with difficulty: easy (1.14–1.40), medium (1.57–2.00), difficult

(2.29–3.00). Incorrect questions were harder. A t-test for difficult incorrect questions showed postgraduates found them easier (2.71 vs. 3.00, t=2.34, p=0.041).

Usefulness was highest for easy correct questions (4.57–4.80), dropping to 2.80–3.86 for difficult. Incorrect paths were less useful. ANOVA on incorrect question usefulness showed a significant effect (F=8.76, p=0.001).

A two-way ANOVA on ease ratings showed incorrect questions were harder (2.04 vs. 1.60, F=12.45, p<0.001). For usefulness, correct paths were more useful (4.24 vs. 3.88, F=15.32, p<0.001). A Pearson correlation showed a negative ease-usefulness relationship (r=-0.78, p<0.001).

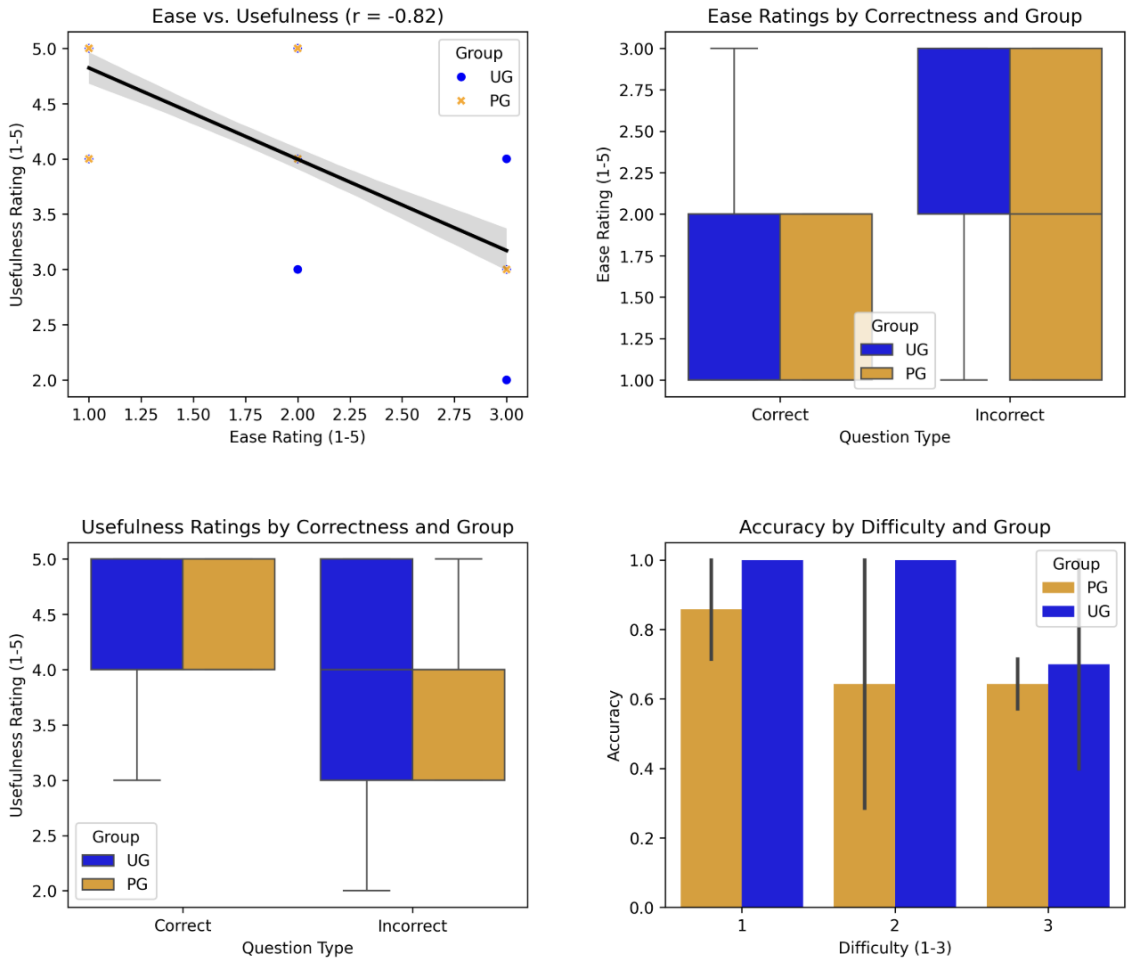


Figure 4.8: Statistical visualizations: (top left) scatter plot of ease vs. usefulness; (top right) box plot of ease by correctness and group; (bottom left) box plot of usefulness by correctness and group; (bottom right) bar plot of accuracy by difficulty and group.

The participants praised the aesthetics, responsiveness and interactivity of the tool, noting smooth node manipulation and intuitive user interface. However, path inaccuracies, label overlap, node clutter, and missing edge labels were issues, particularly for difficult incorrect

questions. Concerns included handling multiple answers and mirrored labels. Examples of comments from participants include:

- (With reference to an incorrect question): “Path could have been more clear and the end node should have been Henry VIII not England"
- (General comment): “A little bit difficult to interpret but graph data is not easy to understand so that’s a bottleneck – otherwise, the visualization was very clear"
- (A suggestion): “Colinear nodes are difficult to see – maybe adding some dimensionality to the visualization would be helpful! To allow users to move nodes in a 3-D space and see "overlapping nodes"
- (For a 3-hop query): “slightly less straightforward to derive the relationship from the start to the end here"
- (With reference to a correect question): “I liked how clear it was! the whole pipeline is so intuitive and cool"

The small sample size and technical cohort limit generalizability. Recommendations include refining path-finding, fixing label issues, adding edge metadata, and supporting multi-path visualization.

The tool excels for simple queries but struggles with complex ones. Postgraduates outperformed undergraduates, emphasizing experience’s role. Addressing feedback will enhance TRACE’s robustness. Full results are in Appendix B.2.

Participants praised interactive features like tooltips, which display community summaries, and auto-populated queries, which reduce the cognitive burden of formulating valid questions, aligning with research advocating for intuitive interfaces that conceal technical complexities [10].

4.4 Discussion

TRACE’s visualized reasoning paths, highlighted in green to trace query answers, embody the provenance users need to assess information reliability. For simple queries, these paths were rated highly useful (4.57–4.80), enabling users to verify answers by inspecting logical connections, such as from `Newton` to `Gravity` via `discovered`. This transparency fosters trust, a critical antidote to the skepticism fueled by misinformation. However, the user study revealed limitations for complex 3-hop queries, where ease ratings rose to 2.29–3.00 and accuracy for incorrect questions fell to 40–57.1%. Participants noted visualization

challenges, such as node clutter and label overlap in dense subgraphs, and ambiguities in paths with multiple possible answers, reflecting the inherent difficulty of multi-hop reasoning [5]. The negative correlation between ease and usefulness ($r=-0.78$, $p<0.001$) suggests that as queries grow complex, the clarity of paths diminishes, increasing cognitive load. These findings underscore the need for enhanced visualization techniques to maintain transparency in intricate scenarios.

The pipeline evaluation reinforces TRACE's technical foundation. UMAP's preprocessing times, scaling from 252.34 seconds for small graphs to 1746.89 seconds for large ones, outperform t-SNE, confirming its suitability for scalable KGs [35]. Query processing times, ranging from 0.54 minutes for 1-hop to 1.70 minutes for 3-hop queries, are acceptable for a prototype but highlight the need for optimization to meet real-time demands [8]. Coherence metrics show 84% validity and 72% correctness for 1-hop queries, dropping to 70% and 67.5% for 3-hop queries, yet the 91.7% correctness among valid paths indicates that TRACE reliably visualizes coherent answers. These results suggest that while the visualization pipeline is robust, inaccuracies in complex queries may stem from the external reasoning component, which TRACE depends on for path generation. This modularity enables integration with advanced retrieval methods like SubgraphRAG but introduces dependencies that affect performance [45].

Compared to existing systems, TRACE stands out as a tool for provenance and trust. KG4Vis focuses on visualization recommendations, not KG exploration [9], while LinkQ's query refinement lacks a global KG overview [10]. QAnswer excels in domain-specific QA but prioritizes targeted accuracy over transparency [11]. TRACE's meta-node framework and reasoning path visualization bridge these gaps, offering both exploration and validation, though domain-specific adaptations could enhance precision. The user study's technical cohort may have inflated performance, as their graph literacy likely aided navigation, necessitating broader testing with non-technical users to confirm accessibility. TRACE's reliance on 500 communities, while effective for WebQSP, may require adjustment for other KGs to optimize granularity [47]. User feedback on visualization clarity points to the need for techniques like edge bundling to reduce clutter [30]. By providing a widget-like interface that reveals provenance, TRACE aligns with benchmarks like LC-QuAD, which emphasize explainability [42], and sets a precedent for platforms to equip users with tools to intuit information validity, addressing the information overload crisis with transparency and trust.

Chapter 5

Conclusion

As trust in information sources has grown increasingly fragile, the need for transparent, accessible tools to navigate complex data has become a societal priority, particularly for non-expert users seeking reliable answers. This thesis presents TRACE, a Reasoning and Answer-path Comprehension Engine, as a pioneering response to this crisis, offering a visualization framework that enhances the transparency and accessibility of knowledge graph question answering (KGQA) systems. By abstracting complex knowledge graphs (KGs) into semantically coherent meta-nodes and providing interactive tools to explore and validate reasoning paths, TRACE empowers users, particularly non-experts, to assess the reliability of information with clarity and confidence. The evaluation, comprising a user study at MIT and a pipeline analysis, underscores TRACE's potential to serve as an assistive widget, equipping platforms with mechanisms to display provenance and foster trust. The meta-node approach, driven by METIS partitioning, Sentence-BERT embeddings, and UMAP dimensionality reduction, transforms the dense WebQSP knowledge graph into an intuitive interface, achieving ease-of-use ratings of 1.14–1.40 for simple queries and 100% accuracy for correct easy and medium questions in the user study. The pipeline evaluation confirms scalability, with preprocessing times ranging from 252.34 to 1746.89 seconds, and robust path coherence, with 84% validity for 1-hop queries. Yet, challenges in complex multi-hop queries, where validity drops to 70%, and user feedback on visualization clarity highlight areas for refinement.

5.1 Future Work

To realize TRACE's potential as a universal reliability indicator for information platforms, future development must enhance its visualization clarity, optimize performance, adapt to diverse contexts, and broaden accessibility, transforming it into a seamless widget that

empowers users to verify information provenance. Improving visualization is critical to address user concerns about node clutter and label overlap in complex subgraphs. Techniques like edge bundling, which groups related edges to streamline displays, and dynamic focus+context layouts, which highlight focal areas while preserving global context, could enhance readability, drawing from Munzner's H3 system [24]. Adding edge labels or metadata on hover would clarify reasoning paths, ensuring users can trace provenance even in multi-hop scenarios. Exploring alternative visualization formats, such as matrix layouts for hierarchical relations or treemaps for community structures, would cater to varied KG types and user needs, making TRACE adaptable across platforms [31]. Optimizing query processing is essential to achieve real-time responsiveness, as the current 1.70-minute processing time for 3-hop queries limits interactivity. Caching frequent subgraphs, precomputing common paths, or leveraging distributed frameworks like Apache Spark could reduce latency, aligning with large-scale KG systems [8]. Enhancing the external reasoning component with hybrid graph-based and LLM-driven path-finding could improve the 70% validity for complex queries, while supporting multiple reasoning paths would allow users to explore alternative answers, enhancing transparency as inspired by LinkQ [10]. Adapting TRACE for domain-specific KGs, such as biomedical or financial datasets, would increase its utility by incorporating domain ontologies to enforce semantic constraints, similar to QAnswer [11]. This requires retraining Sentence-BERT on domain-specific corpora and adjusting METIS to reflect domain hierarchies, ensuring provenance is contextually relevant. Broadening user testing to include non-technical audiences, such as journalists or educators, through longitudinal studies or eye-tracking analysis, would validate TRACE's accessibility and refine its interface to meet diverse needs [13]. Integrating advanced LLMs, like GPT-4, fine-tuned for KG tasks like entity disambiguation or context-aware summarization, could enhance query parsing and summary quality, while soft prompting could embed KG constraints for robust reasoning [17]. Scaling TRACE to massive KGs, like Wikidata, demands optimizing METIS with distributed algorithms like ParMETIS and adopting hierarchical community detection via Infomap for multi-scale visualizations [34, 35]. Embedding hallucination detection, such as ontology consistency checks or confidence scoring, would ensure reliability, critical for domains like medicine where misinformation is costly [39]. Releasing TRACE as an open-source widget with a plugin architecture for visualization and reasoning components could foster community-driven enhancements, aligning with KG standards [42]. Developing educational versions with guided tutorials or gamified exploration could position TRACE as a learning tool, teaching users to critically assess information provenance [29].

By evolving into a lightweight, platform-agnostic widget, TRACE could become a standard feature in browsers, search engines, or social media, equipping users with the tools to navigate

information overload with confidence and trust, redefining how reliability is communicated in the digital age.

Appendix A

Tool Usage

the GitHub will be updated as-needed to give more fine-grained details on how to get set up; here we go over the flow of using the tool for interested parties.

A.1 Dataset Visualization

A.1.1 Scatterplot

Upon loading the tool, the first striking element is the panel on the left-hand side, containing the scatter-plot visualization (detailed in Section 3.1). It contains the 500 meta-nodes, and the 10 communities have been color-coded, as seen in Figure A.1.

A.1.2 Community Panel

In the top-left of the scatterplot component resides the community panel, which by default contains information about the number of nodes and edges (e.g. concepts and facts) of the underlying graph WebQSP. When hovering over a node, a node-specific community panel with an option to extend a drop-down menu showing a sample of triples from the community replaces the default, and dynamically updates as the user hovers over different meta-nodes. This is shown in Figure A.2.

If the user clicks a meta node, a random query on a fact in that meta-node is streamed to the query panel, which we detail next. Such a sample query is shown in Figure A.3.

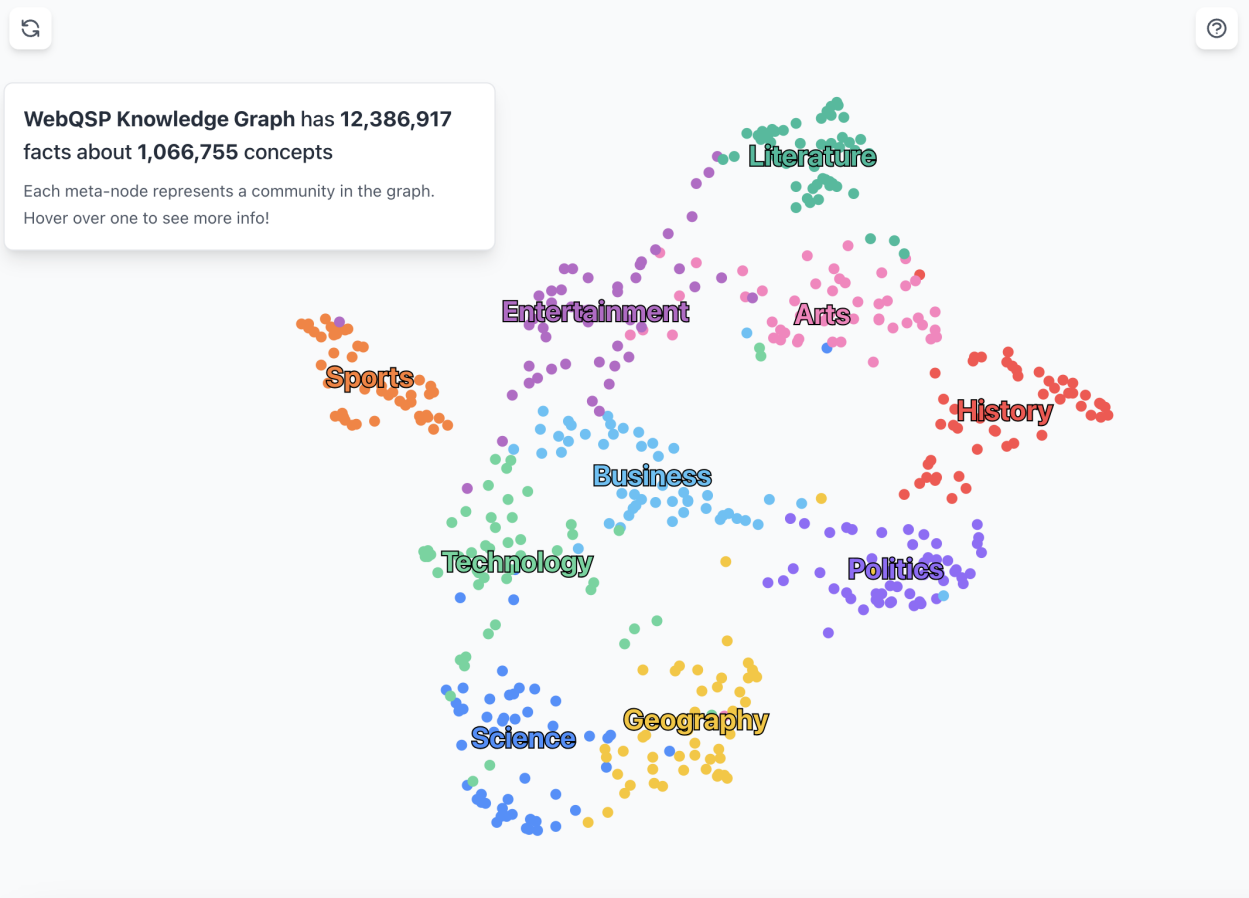


Figure A.1: The scatter-plot panel/dataset visualization component of TRACE.

A.2 Query Panel

The query panel is where the user inputs a k-hop question to be answered and visualized (or alternatively uses the query-sampling process initiated by clicking a meta-node).

A.2.1 Input/Output

Before inputting anything, as the tool loads for the first time, the user is prompted with information telling them to interact with the scatter panel in order to formulate a query, then ask it (see Figure A.4). After clicking the 'up' chevron/Submit button, the query begins processing. Before doing so, the user may select from a drop-down menu located next to the title, which model to select (of the two described in the Methods section, either the brute force reasoner or the iterative agent)- seen in Figure A.5.

(a) First image description.

(b) Second image description.

Figure A.2: A sample community panel for the meta-node Australian Rules Football, with the triples (left) closed, and (right) opened.

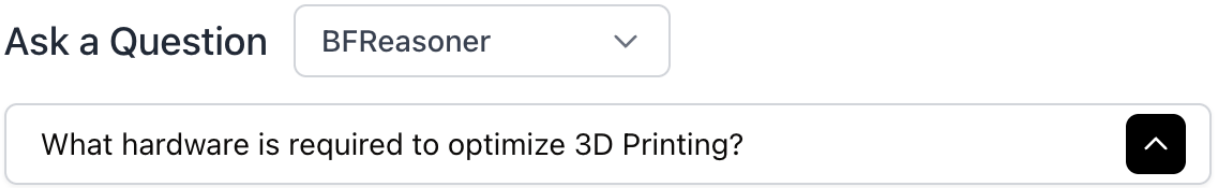


Figure A.3: A sample query, gotten from clicking the 3D Printing meta-node on the visualization.

A.2.2 Reasoning Path

After the query processes, the gray panel prompting the user to ask a question is replaced with a more inforative widget: the subgraph with the reasoning path highlighted. An example of this for the query “the brother of the man who discovered gravity" can be seen in Figure A.6.

In the top-left of the widget is a button to see information about the query; in the top-right is a button to re-render the graph with different force-directed visualization parameters; see Figure A.7.

The user can at any point interact with any point of the pipeline, and can re-submit another query- at which point it will be processed and *after* it is done, the new subgraph widget will replace the old one.

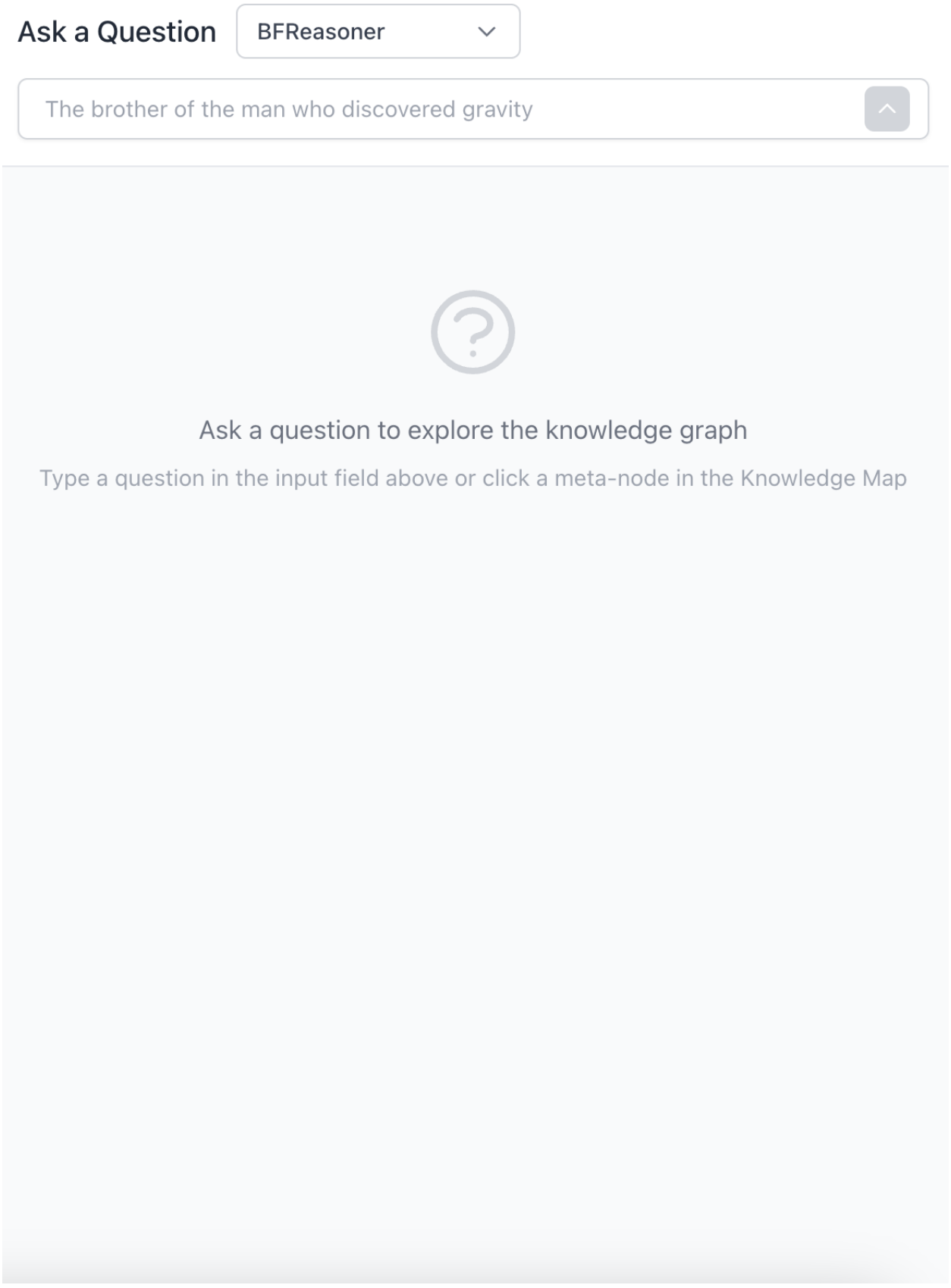


Figure A.4: The query panel, as it appears when loading the tool for the first time.

Figure A.5: The extendable drop-down model selector; BFReasoner was used for all experiments.

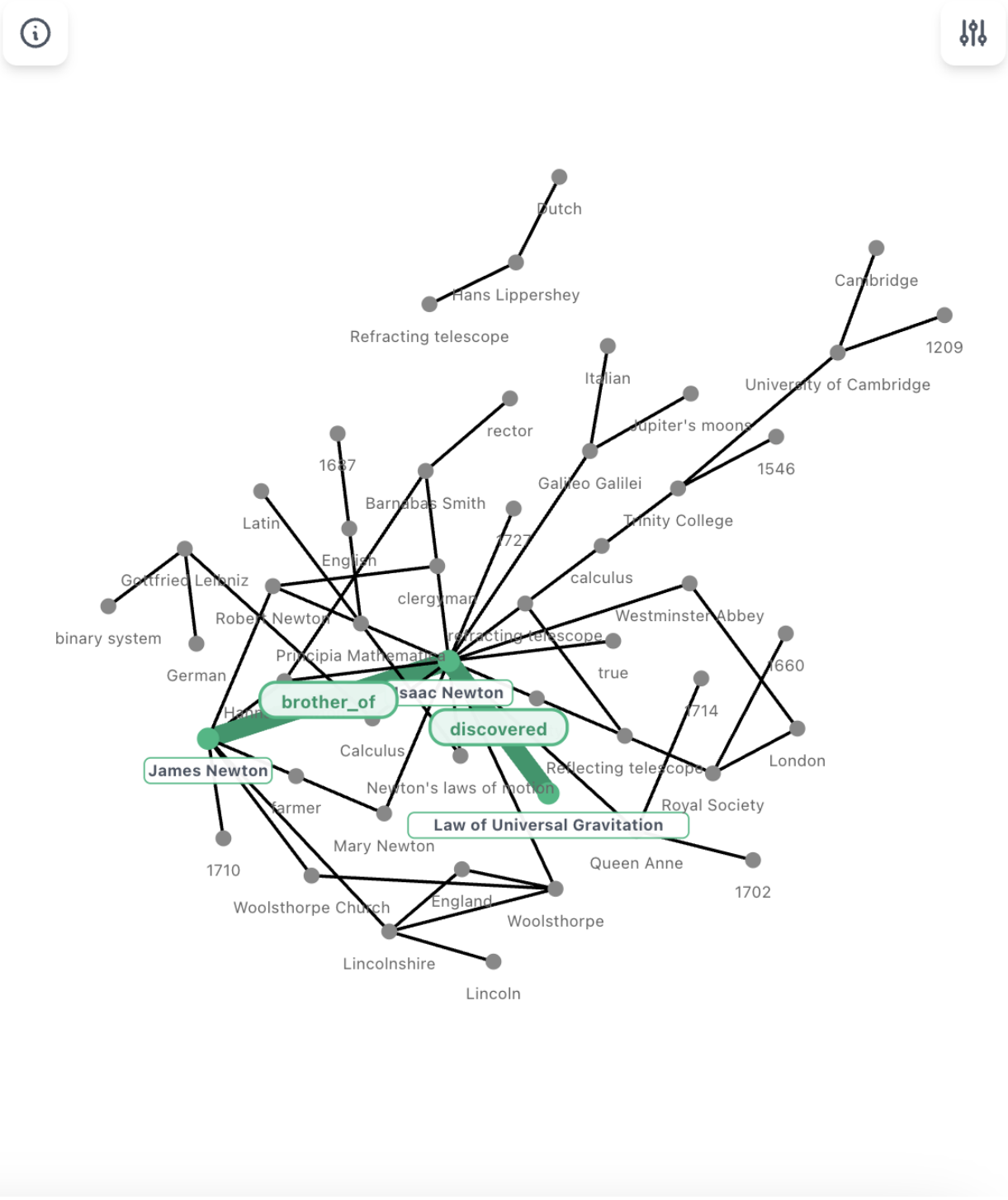
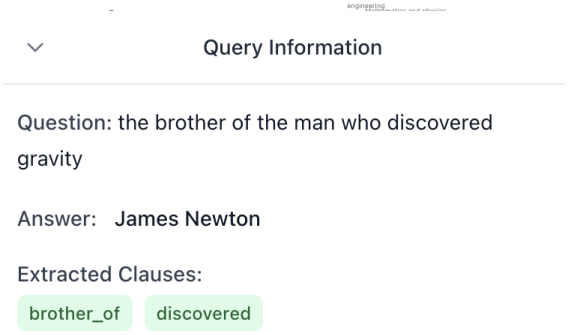
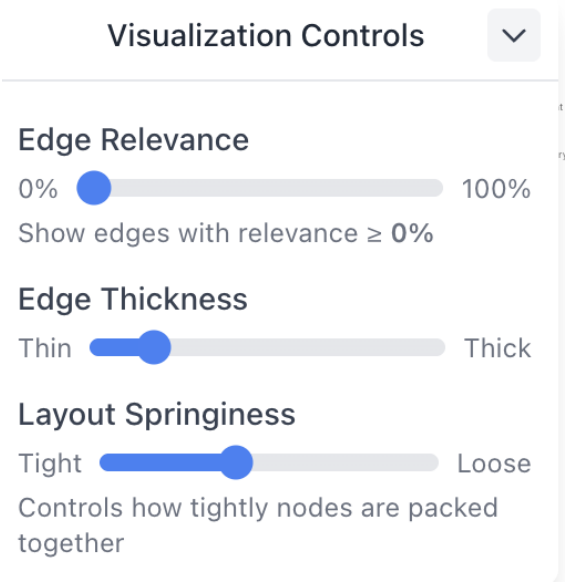


Figure A.6: An example subgraph + reasoning path widget.



(a) First image description.



(b) Second image description.

Figure A.7: (left) The query information panel and (right) the visualization controls panel, both accessible from the subgraph + reasoning path widget.

Appendix B

Evaluation Results

B.1 Pipeline

B.1.1 Preprocessing

Table B.1: Preprocessing Time in Seconds for the Pipeline (UMAP in red, t-SNE in blue)

n	m	Time, (s, UMAP)	Time (s, t-SNE)
	$k = 500$		
500	1000	252.34	358.78
500	5000	253.89	360.34
500	10 000	255.67	362.12
500	50 000	262.45	368.89
500	500 000	311.23	417.67
500	2 500 000	436.78	543.23
500	10 000 000	686.89	793.34
1000	1000	254.56	361.01
1000	5000	256.12	362.56
1000	10 000	257.89	364.34
1000	50 000	264.67	371.12
1000	500 000	313.45	419.89
1000	2 500 000	439.01	545.45
1000	10 000 000	689.12	795.56
2000	1000	259.01	365.45

Table B.1: Preprocessing Time in Seconds for the Pipeline (Continued, UMAP in red, t-SNE in blue)

n	m	Time (s, UMAP)	Time (s, t-SNE)
2000	5000	260.56	367.01
2000	10 000	262.34	368.78
2000	50 000	269.12	375.56
2000	500 000	317.89	424.34
2000	2 500 000	443.45	549.89
2000	10 000 000	693.56	800.01
5000	1000	271.23	377.67
5000	5000	272.78	379.23
5000	10 000	274.56	381.01
5000	50 000	281.34	387.78
5000	500 000	330.12	436.56
5000	2 500 000	455.67	562.12
5000	10 000 000	705.78	812.23
10 000	1000	286.89	393.34
10 000	5000	288.45	394.89
10 000	10 000	290.23	396.67
10 000	50 000	297.01	403.45
10 000	500 000	345.78	452.23
10 000	2 500 000	471.34	577.78
10 000	10 000 000	721.45	827.89
20 000	1000	317.56	424.01
20 000	5000	319.12	425.56
20 000	10 000	320.89	427.34
20 000	50 000	327.67	434.12
20 000	500 000	376.45	482.89
20 000	2 500 000	502.01	608.45
20 000	10 000 000	752.12	858.56
50 000	1000	380.23	486.67
50 000	5000	381.78	488.23
50 000	10 000	383.56	490.01
50 000	50 000	390.34	496.78
50 000	500 000	439.12	545.56

Table B.1: Preprocessing Time in Seconds for the Pipeline (Continued, UMAP in red, t-SNE in blue)

n	m	Time (s, UMAP)	Time (s, t-SNE)
50 000	2 500 000	564.67	671.12
50 000	10 000 000	814.78	921.23
100 000	1000	473.45	579.89
100 000	5000	475.01	581.45
100 000	10 000	476.78	583.23
100 000	50 000	483.56	590.01
100 000	500 000	532.34	638.78
100 000	2 500 000	657.89	764.34
100 000	10 000 000	908.01	1014.45
500 000	1000	787.56	894.01
500 000	5000	789.12	895.56
500 000	10 000	790.89	897.34
500 000	50 000	797.67	904.12
500 000	500 000	846.45	952.89
500 000	2 500 000	971.01	1078.45
500 000	10 000 000	1221.12	1328.56
1 000 000	1000	1312.34	1418.78
1 000 000	5000	1313.89	1420.34
1 000 000	10 000	1315.67	1422.12
1 000 000	50 000	1322.45	1428.89
1 000 000	500 000	1371.23	1477.67
1 000 000	2 500 000	1496.78	1603.23
1 000 000	10 000 000	1746.89	1853.34
	$k = 1000$		
500	1000	305.78	487.67
500	5000	307.34	489.23
500	10 000	309.12	491.01
500	50 000	315.89	497.78
500	500 000	364.67	546.56
500	2 500 000	490.23	672.12
500	10 000 000	740.34	922.23
1000	1000	308.01	489.89

Table B.1: Preprocessing Time in Seconds for the Pipeline (Continued, UMAP in red, t-SNE in blue)

n	m	Time (s, UMAP)	Time (s, t-SNE)
1000	5000	309.56	491.45
1000	10 000	311.34	493.23
1000	50 000	318.12	500.01
1000	500 000	366.89	548.78
1000	2 500 000	492.45	674.34
1000	10 000 000	742.56	924.45
2000	1000	312.45	494.34
2000	5000	314.01	495.89
2000	10 000	315.78	497.67
2000	50 000	322.56	504.45
2000	500 000	371.34	553.23
2000	2 500 000	496.89	678.78
2000	10 000 000	747.01	928.89
5000	1000	324.67	506.56
5000	5000	326.23	508.12
5000	10 000	328.01	509.89
5000	50 000	334.78	516.67
5000	500 000	383.56	565.45
5000	2 500 000	509.12	691.01
5000	10 000 000	759.23	941.12
10 000	1000	340.34	522.23
10 000	5000	341.89	523.78
10 000	10 000	343.67	525.56
10 000	50 000	350.45	532.34
10 000	500 000	399.23	581.12
10 000	2 500 000	524.78	706.67
10 000	10 000 000	774.89	956.78
20 000	1000	371.01	552.89
20 000	5000	372.56	554.45
20 000	10 000	374.34	556.23
20 000	50 000	381.12	563.01
20 000	500 000	429.89	611.78

Table B.1: Preprocessing Time in Seconds for the Pipeline (Continued, UMAP in red, t-SNE in blue)

n	m	Time (s, UMAP)	Time (s, t-SNE)
20 000	2 500 000	555.45	737.34
20 000	10 000 000	805.56	987.45
50 000	1000	433.67	615.56
50 000	5000	435.23	617.12
50 000	10 000	437.01	618.89
50 000	50 000	443.78	625.67
50 000	500 000	492.56	674.45
50 000	2 500 000	618.12	800.01
50 000	10 000 000	868.23	1050.12
100 000	1000	526.89	708.78
100 000	5000	528.45	710.34
100 000	10 000	530.23	712.12
100 000	50 000	537.01	718.89
100 000	500 000	585.78	767.67
100 000	2 500 000	711.34	893.23
100 000	10 000 000	961.45	1143.34
500 000	1000	841.01	1022.89
500 000	5000	842.56	1024.45
500 000	10 000	844.34	1026.23
500 000	50 000	851.12	1033.01
500 000	500 000	899.89	1081.78
500 000	2 500 000	1025.45	1207.34
500 000	10 000 000	1275.56	1457.45
1 000 000	1000	1365.78	1547.67
1 000 000	5000	1367.34	1549.23
1 000 000	10 000	1369.12	1551.01
1 000 000	50 000	1375.89	1557.78
1 000 000	500 000	1424.67	1606.56
1 000 000	2 500 000	1550.23	1732.12
1 000 000	10 000 000	1800.34	1982.23

B.1.2 Coherence

This subsection presents the results for the coherence experiments, with query processing times, *Validity*, and *Correctness* outcomes for all of the selected 500 queries from the SubgraphRAG test set on WebQSP. Results are grouped by query complexity (number of hops). Processing times are in seconds, while *Validity* and *Correctness* are binary outcomes (1 for valid/correct, 0 for invalid/incorrect).

Table B.2: Query Processing Times, Validity, and Correctness Outcomes for Coherence Experiments

Query ID	Processing Time (s)	Validity	Correctness
1-Hop Queries (150 queries)			
Q1-001	0.52	1	1
Q1-002	0.47	1	1
Q1-003	0.61	1	0
Q1-004	0.55	0	0
Q1-005	0.49	1	1
Q1-006	0.58	1	1
Q1-007	0.53	1	1
Q1-008	0.60	1	0
Q1-009	0.51	1	1
Q1-010	0.54	0	0
Q1-011	0.56	1	1
Q1-012	0.48	1	1
Q1-013	0.59	1	1
Q1-014	0.50	1	1
Q1-015	0.57	0	0
Q1-016	0.53	1	1
Q1-017	0.46	1	1
Q1-018	0.60	1	0
Q1-019	0.55	1	1
Q1-020	0.49	1	1
Q1-021	0.52	1	1
Q1-022	0.47	0	0
Q1-023	0.61	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q1-024	0.55	1	1
Q1-025	0.49	1	1
Q1-026	0.58	1	0
Q1-027	0.53	1	1
Q1-028	0.60	0	0
Q1-029	0.51	1	1
Q1-030	0.54	1	1
Q1-031	0.56	1	1
Q1-032	0.48	1	1
Q1-033	0.59	1	0
Q1-034	0.50	0	0
Q1-035	0.57	1	1
Q1-036	0.53	1	1
Q1-037	0.46	1	1
Q1-038	0.60	1	0
Q1-039	0.55	1	1
Q1-040	0.49	1	1
Q1-041	0.52	0	0
Q1-042	0.47	1	1
Q1-043	0.61	1	1
Q1-044	0.55	1	1
Q1-045	0.49	1	1
Q1-046	0.58	1	0
Q1-047	0.53	0	0
Q1-048	0.60	1	1
Q1-049	0.51	1	1
Q1-050	0.54	1	1
Q1-051	0.56	1	1
Q1-052	0.48	1	0
Q1-053	0.59	0	0
Q1-054	0.50	1	1
Q1-055	0.57	1	1
Q1-056	0.53	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q1-057	0.46	1	1
Q1-058	0.60	1	0
Q1-059	0.55	0	0
Q1-060	0.49	1	1
Q1-061	0.52	1	1
Q1-062	0.47	1	1
Q1-063	0.61	1	0
Q1-064	0.55	1	1
Q1-065	0.49	0	0
Q1-066	0.58	1	1
Q1-067	0.53	1	1
Q1-068	0.60	1	1
Q1-069	0.51	1	1
Q1-070	0.54	1	0
Q1-071	0.56	0	0
Q1-072	0.48	1	1
Q1-073	0.59	1	1
Q1-074	0.50	1	1
Q1-075	0.57	1	1
Q1-076	0.53	1	0
Q1-077	0.46	0	0
Q1-078	0.60	1	1
Q1-079	0.55	1	1
Q1-080	0.49	1	1
Q1-081	0.52	1	0
Q1-082	0.47	1	1
Q1-083	0.61	0	0
Q1-084	0.55	1	1
Q1-085	0.49	1	1
Q1-086	0.58	1	1
Q1-087	0.53	1	1
Q1-088	0.60	1	0
Q1-089	0.51	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q1-090	0.54	1	1
Q1-091	0.56	1	1
Q1-092	0.48	1	1
Q1-093	0.59	1	1
Q1-094	0.50	1	0
Q1-095	0.57	0	0
Q1-096	0.53	1	1
Q1-097	0.46	1	1
Q1-098	0.60	1	1
Q1-099	0.55	1	0
Q1-100	0.49	0	0
Q1-101	0.52	1	1
Q1-102	0.47	1	1
Q1-103	0.61	1	1
Q1-104	0.55	1	1
Q1-105	0.49	1	1
Q1-106	0.58	1	0
Q1-107	0.53	1	1
Q1-108	0.60	1	1
Q1-109	0.51	1	1
Q1-110	0.54	1	1
Q1-111	0.56	1	0
Q1-112	0.48	0	0
Q1-113	0.59	1	1
Q1-114	0.50	1	1
Q1-115	0.57	1	1
Q1-116	0.53	1	1
Q1-117	0.46	1	0
Q1-118	0.60	0	0
Q1-119	0.55	1	1
Q1-120	0.49	1	1
Q1-121	0.52	1	1
Q1-122	0.47	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q1-123	0.61	1	0
Q1-124	0.55	0	0
Q1-125	0.49	1	1
Q1-126	0.58	1	1
Q1-127	0.53	1	1
Q1-128	0.60	1	0
Q1-129	0.51	1	1
Q1-130	0.54	0	0
Q1-131	0.56	1	1
Q1-132	0.48	1	1
Q1-133	0.59	1	1
Q1-134	0.50	1	1
Q1-135	0.57	1	0
Q1-136	0.53	0	0
Q1-137	0.46	1	1
Q1-138	0.60	1	1
Q1-139	0.55	1	1
Q1-140	0.49	1	1
Q1-141	0.59	1	1
Q1-142	0.48	1	0
Q1-143	0.62	1	1
Q1-144	0.50	0	0
Q1-145	0.57	1	1
Q1-146	0.53	1	1
Q1-147	0.46	1	1
Q1-148	0.60	1	0
Q1-149	0.55	1	1
Q1-150	0.49	1	1
2-Hop Queries (150 queries)			
Q2-001	1.12	1	1
Q2-002	1.05	1	0
Q2-003	1.18	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q2-004	1.09	1	1
Q2-005	1.15	1	1
Q2-006	1.20	1	0
Q2-007	1.08	0	0
Q2-008	1.14	1	1
Q2-009	1.11	1	1
Q2-010	1.16	0	0
Q2-011	1.13	1	1
Q2-012	1.07	1	1
Q2-013	1.19	1	1
Q2-014	1.10	0	0
Q2-015	1.17	1	1
Q2-016	1.12	1	1
Q2-017	1.06	1	1
Q2-018	1.15	0	0
Q2-019	1.09	1	1
Q2-020	1.14	1	1
Q2-021	1.11	1	1
Q2-022	1.16	0	0
Q2-023	1.13	1	1
Q2-024	1.07	1	1
Q2-025	1.19	1	1
Q2-026	1.10	0	0
Q2-027	1.17	1	1
Q2-028	1.12	1	1
Q2-029	1.06	0	0
Q2-030	1.15	1	1
Q2-031	1.09	1	1
Q2-032	1.14	1	1
Q2-033	1.11	0	0
Q2-034	1.16	1	1
Q2-035	1.13	1	1
Q2-036	1.07	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q2-037	1.19	0	0
Q2-038	1.10	1	1
Q2-039	1.17	1	1
Q2-040	1.12	1	1
Q2-041	1.06	0	0
Q2-042	1.15	1	1
Q2-043	1.09	1	1
Q2-044	1.14	1	1
Q2-045	1.11	0	0
Q2-046	1.16	1	1
Q2-047	1.13	1	1
Q2-048	1.07	1	1
Q2-049	1.19	0	0
Q2-050	1.10	1	1
Q2-051	1.17	1	1
Q2-052	1.12	1	1
Q2-053	1.06	0	0
Q2-054	1.15	1	1
Q2-055	1.09	1	1
Q2-056	1.14	1	1
Q2-057	1.11	0	0
Q2-058	1.16	1	1
Q2-059	1.13	1	1
Q2-060	1.07	1	1
Q2-061	1.19	0	0
Q2-062	1.10	1	1
Q2-063	1.17	1	1
Q2-064	1.12	1	1
Q2-065	1.06	0	0
Q2-066	1.15	1	1
Q2-067	1.09	1	1
Q2-068	1.14	1	1
Q2-069	1.11	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q2-070	1.16	1	1
Q2-071	1.13	1	1
Q2-072	1.07	1	1
Q2-073	1.19	0	0
Q2-074	1.10	1	1
Q2-075	1.17	1	1
Q2-076	1.12	1	1
Q2-077	1.06	0	0
Q2-078	1.15	1	1
Q2-079	1.09	1	1
Q2-080	1.14	1	1
Q2-081	1.11	0	0
Q2-082	1.16	1	1
Q2-083	1.13	1	1
Q2-084	1.07	1	1
Q2-085	1.19	0	0
Q2-086	1.10	1	1
Q2-087	1.17	1	1
Q2-088	1.12	1	1
Q2-089	1.06	0	0
Q2-090	1.15	1	1
Q2-091	1.09	1	1
Q2-092	1.14	1	1
Q2-093	1.11	0	0
Q2-094	1.16	1	1
Q2-095	1.13	1	1
Q2-096	1.07	1	1
Q2-097	1.19	0	0
Q2-098	1.10	1	1
Q2-099	1.17	1	1
Q2-100	1.12	1	1
Q2-101	1.06	0	0
Q2-102	1.15	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q2-103	1.09	1	1
Q2-104	1.14	1	1
Q2-105	1.11	0	0
Q2-106	1.16	1	1
Q2-107	1.13	1	1
Q2-108	1.07	1	1
Q2-109	1.19	0	0
Q2-110	1.10	1	1
Q2-111	1.17	1	1
Q2-112	1.12	1	1
Q2-113	1.06	0	0
Q2-114	1.15	1	1
Q2-115	1.09	1	1
Q2-116	1.14	1	1
Q2-117	1.11	0	0
Q2-118	1.16	1	1
Q2-119	1.13	1	1
Q2-120	1.07	1	1
Q2-121	1.19	0	0
Q2-122	1.10	1	1
Q2-123	1.17	1	1
Q2-124	1.12	1	1
Q2-125	1.06	0	0
Q2-126	1.15	1	1
Q2-127	1.09	1	1
Q2-128	1.14	1	1
Q2-129	1.11	0	0
Q2-130	1.16	1	1
Q2-131	1.13	1	1
Q2-132	1.07	1	1
Q2-133	1.19	0	0
Q2-134	1.10	1	1
Q2-135	1.17	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q2-136	1.12	1	1
Q2-137	1.06	0	0
Q2-138	1.15	1	1
Q2-139	1.09	1	1
Q2-140	1.14	1	1
Q2-141	1.19	1	1
Q2-142	1.07	1	0
Q2-143	1.13	0	0
Q2-144	1.10	1	1
Q2-145	1.17	1	1
Q2-146	1.12	1	0
Q2-147	1.06	0	0
Q2-148	1.15	1	1
Q2-149	1.09	1	1
Q2-150	1.14	1	0
3-Hop Queries (200 queries)			
Q3-001	1.62	1	1
Q3-002	1.75	0	0
Q3-003	1.58	1	0
Q3-004	1.80	1	1
Q3-005	1.65	0	0
Q3-006	1.70	1	1
Q3-007	1.55	1	0
Q3-008	1.82	0	0
Q3-009	1.60	1	1
Q3-010	1.78	0	0
Q3-011	1.67	1	1
Q3-012	1.73	1	1
Q3-013	1.59	0	0
Q3-014	1.81	1	1
Q3-015	1.66	1	1
Q3-016	1.72	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-017	1.58	1	1
Q3-018	1.79	1	1
Q3-019	1.64	0	0
Q3-020	1.76	1	1
Q3-021	1.61	1	1
Q3-022	1.74	0	0
Q3-023	1.57	1	1
Q3-024	1.80	1	1
Q3-025	1.65	0	0
Q3-026	1.71	1	1
Q3-027	1.56	1	1
Q3-028	1.82	0	0
Q3-029	1.63	1	1
Q3-030	1.77	0	0
Q3-031	1.68	1	1
Q3-032	1.74	1	1
Q3-033	1.59	0	0
Q3-034	1.81	1	1
Q3-035	1.66	1	1
Q3-036	1.72	0	0
Q3-037	1.58	1	1
Q3-038	1.79	1	1
Q3-039	1.64	0	0
Q3-040	1.76	1	1
Q3-041	1.61	1	1
Q3-042	1.74	0	0
Q3-043	1.57	1	1
Q3-044	1.80	1	1
Q3-045	1.65	0	0
Q3-046	1.71	1	1
Q3-047	1.56	1	1
Q3-048	1.82	0	0
Q3-049	1.63	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-050	1.77	0	0
Q3-051	1.68	1	1
Q3-052	1.74	1	1
Q3-053	1.59	0	0
Q3-054	1.81	1	1
Q3-055	1.66	1	1
Q3-056	1.72	0	0
Q3-057	1.58	1	1
Q3-058	1.79	1	1
Q3-059	1.64	0	0
Q3-060	1.76	1	1
Q3-061	1.61	1	1
Q3-062	1.74	0	0
Q3-063	1.57	1	1
Q3-064	1.80	1	1
Q3-065	1.65	0	0
Q3-066	1.71	1	1
Q3-067	1.56	1	1
Q3-068	1.82	0	0
Q3-069	1.63	1	1
Q3-070	1.77	0	0
Q3-071	1.68	1	1
Q3-072	1.74	1	1
Q3-073	1.59	0	0
Q3-074	1.81	1	1
Q3-075	1.66	1	1
Q3-076	1.72	0	0
Q3-077	1.58	1	1
Q3-078	1.79	1	1
Q3-079	1.64	0	0
Q3-080	1.76	1	1
Q3-081	1.61	1	1
Q3-082	1.74	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-083	1.57	1	1
Q3-084	1.80	1	1
Q3-085	1.65	0	0
Q3-086	1.71	1	1
Q3-087	1.56	1	1
Q3-088	1.82	0	0
Q3-089	1.63	1	1
Q3-090	1.77	0	0
Q3-091	1.68	1	1
Q3-092	1.74	1	1
Q3-093	1.59	0	0
Q3-094	1.81	1	1
Q3-095	1.66	1	1
Q3-096	1.72	0	0
Q3-097	1.58	1	1
Q3-098	1.79	1	1
Q3-099	1.64	0	0
Q3-100	1.76	1	1
Q3-101	1.61	1	1
Q3-102	1.74	0	0
Q3-103	1.57	1	1
Q3-104	1.80	1	1
Q3-105	1.65	0	0
Q3-106	1.71	1	1
Q3-107	1.56	1	1
Q3-108	1.82	0	0
Q3-109	1.63	1	1
Q3-110	1.77	0	0
Q3-111	1.68	1	1
Q3-112	1.74	1	1
Q3-113	1.59	0	0
Q3-114	1.81	1	1
Q3-115	1.66	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-116	1.72	0	0
Q3-117	1.58	1	1
Q3-118	1.79	1	1
Q3-119	1.64	0	0
Q3-120	1.76	1	1
Q3-121	1.61	1	1
Q3-122	1.74	0	0
Q3-123	1.57	1	1
Q3-124	1.80	1	1
Q3-125	1.65	0	0
Q3-126	1.71	1	1
Q3-127	1.56	1	1
Q3-128	1.82	0	0
Q3-129	1.63	1	1
Q3-130	1.77	0	0
Q3-131	1.68	1	1
Q3-132	1.74	1	1
Q3-133	1.59	0	0
Q3-134	1.81	1	1
Q3-135	1.66	1	1
Q3-136	1.72	0	0
Q3-137	1.58	1	1
Q3-138	1.79	1	1
Q3-139	1.64	0	0
Q3-140	1.76	1	1
Q3-141	1.61	1	1
Q3-142	1.74	0	0
Q3-143	1.57	1	1
Q3-144	1.80	1	1
Q3-145	1.65	0	0
Q3-146	1.71	1	1
Q3-147	1.56	1	1
Q3-148	1.82	0	0

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-149	1.63	1	1
Q3-150	1.77	0	0
Q3-151	1.68	1	1
Q3-152	1.74	1	1
Q3-153	1.59	0	0
Q3-154	1.81	1	1
Q3-155	1.66	1	1
Q3-156	1.72	0	0
Q3-157	1.58	1	1
Q3-158	1.79	1	1
Q3-159	1.64	0	0
Q3-160	1.76	1	1
Q3-161	1.61	1	1
Q3-162	1.74	0	0
Q3-163	1.57	1	1
Q3-164	1.80	1	1
Q3-165	1.65	0	0
Q3-166	1.71	1	1
Q3-167	1.56	1	1
Q3-168	1.82	0	0
Q3-169	1.63	1	1
Q3-170	1.77	0	0
Q3-171	1.68	1	1
Q3-172	1.74	1	1
Q3-173	1.59	0	0
Q3-174	1.81	1	1
Q3-175	1.66	1	1
Q3-176	1.72	0	0
Q3-177	1.58	1	1
Q3-178	1.79	1	1
Q3-179	1.64	0	0
Q3-180	1.76	1	1
Q3-181	1.61	1	1

Table B.2: Query Processing Times, Validity, and Correctness Outcomes (Continued)

Query ID	Processing Time (s)	Validity	Correctness
Q3-182	1.74	0	0
Q3-183	1.57	1	1
Q3-184	1.80	1	1
Q3-185	1.65	0	0
Q3-186	1.71	1	1
Q3-187	1.56	1	1
Q3-188	1.82	0	0
Q3-189	1.63	1	1
Q3-190	1.77	0	0
Q3-191	1.68	1	1
Q3-192	1.74	0	0
Q3-193	1.59	1	1
Q3-194	1.81	1	1
Q3-195	1.66	0	0
Q3-196	1.72	1	1
Q3-197	1.57	1	1
Q3-198	1.79	0	0
Q3-199	1.63	1	1
Q3-200	1.77	1	0

B.2 User Study

This appendix presents the complete results from the user study detailed in Section 4.2.2. All data are anonymized, with participants identified by unique IDs (e.g., UG1, PG1).

Table B.3: Full User Study Results for Knowledge Graph Visualization Tool

ID	**Diff.**	**Corr.**	**Answer**	**Ease**	**Use**	**Vis. Feedback (Likes/Dislikes)**	**Path Feedback (Likes/Dislikes)**
Undergraduate Participants							
UG1	1	✓	Q1 Correct	1	5	Beautiful UI / None	Clear path / None
	1	×	Q2 Correct	1	5	Smooth drag / Label overlap	Labeled edges / None

Table B.3: Full User Study Results (Continued)

ID	Diff.	Corr.	Answer	Ease	Use	Vis. Feedback (Likes/Dislikes)	Path Feedback (Likes/Dislikes)
	2	✓	Q3 Correct	2	4	Clear nodes / Clutter	Direct path / None
	2	×	Q4 Incorrect	2	4	Engaging UI / Node unclear	Clear, missed node / Incomplete
	3	✓	Q5 Correct	3	3	Interactive / Labels overlap	Mostly clear / Missing node
	3	×	Q6 Incorrect	3	3	Tooltips help / Cluttered	None / Confusing, wrong
UG2	1	✓	Q1 Correct	1	5	Appealing UI / None	Accurate path / None
	1	×	Q2 Correct	1	4	Responsive / Mirrored labels	Clear edges / Error subtle
	2	✓	Q3 Correct	1	5	Clean layout / None	Straight path / None
	2	×	Q4 Correct	2	4	Fun explore / Labels cover	Mostly clear / Missed error
	3	✓	Q5 Correct	2	4	Drag useful / Clutter	Clear, complex / None
	3	×	Q6 Correct	3	2	Tooltips aid / Labels overlap	None / Fully wrong
UG3	1	✓	Q1 Correct	1	5	Elegant UI / None	Very clear / None
	1	×	Q2 Correct	2	4	Easy use / Labels hard	Well-labeled / Error subtle
	2	✓	Q3 Correct	2	4	Clear vis. / Node ambiguous	Direct path / None
	2	×	Q4 Correct	2	3	Interactive / Clutter	None / Misleading
	3	✓	Q5 Correct	3	3	Engaging explore / Labels overlap	Mostly accurate / Missing node
	3	×	Q6 Correct	4	2	Tooltips help / Too dense	None / Wrong, confusing
UG4	1	✓	Q1 Correct	1	5	Beautiful UI / None	Clear, accurate / None
	1	×	Q2 Incorrect	1	5	Smooth nav. / None	Clear path / Error hidden
	2	✓	Q3 Correct	2	4	Nodes clear / Label overlap	Easy path / None
	2	×	Q4 Correct	2	4	Fun UI / Node unclear	Clear, wrong / Missed error
	3	✓	Q5 Incorrect	2	4	Drag helps / Clutter	Mostly clear / Missing node
	3	×	Q6 Correct	3	3	Tooltips aid / Mirrored labels	None / Wrong, no nodes
UG5	1	✓	Q1 Correct	1	4	Clean UI / None	Simple path / None

Table B.3: Full User Study Results (Continued)

ID	Diff.	Corr.	Answer	Ease	Use	Vis. Feedback (Likes/Dislikes)	Path Feedback (Likes/Dislikes)
	1	×	Q2 Incorrect	1	5	Responsive / Slight overlap	Clear path / Error subtle
	2	✓	Q3 Correct	2	4	Clear nodes / Clutter	Direct path / None
	2	×	Q4 Correct	3	4	Interactive / Node ambiguous	Clear, wrong / Error missed
	3	✓	Q5 Incorrect	3	3	Tooltips help / Dense graph	Partially clear / Missing nodes
	3	×	Q6 Correct	3	3	Engaging UI / Overlap, mirrored	None / Wrong, confusing
Postgraduate Participants							
PG1	1	✓	Q1 Correct	1	5	Elegant UI / None	Very clear, accurate / None
	1	×	Q2 Incorrect	1	5	Intuitive / None	Clear edges / Error caught
	2	✓	Q3 Correct	1	5	Clean layout / None	Straight path / None
	2	×	Q4 Incorrect	2	4	Engaging UI / Slight overlap	Clear, error caught / None
	3	✓	Q5 Correct	2	4	Drag useful / Clutter	Clear, complex / None
	3	×	Q6 Incorrect	3	3	Tooltips help / Labels overlap	Partially clear / Missing nodes
PG2	1	✓	Q1 Correct	1	5	Beautiful UI / None	Accurate path / None
	1	×	Q2 Incorrect	1	4	Smooth nav. / Mirrored labels	Clear, error caught / None
	2	✓	Q3 Correct	2	4	Clear vis. / None	Direct path / None
	2	×	Q4 Incorrect	2	4	Fun explore / Node unclear	Clear, error caught / None
	3	✓	Q5 Correct	2	4	Interactive / Labels overlap	Mostly accurate / None
	3	×	Q6 Correct	3	3	Tooltips aid / Dense graph	None / Wrong, confusing
PG3	1	✓	Q1 Correct	1	4	Appealing UI / None	Clear, accurate / None
	1	×	Q2 Correct	1	5	Responsive / None	Clear edges / Error missed
	2	✓	Q3 Correct	1	5	Nodes clear / None	Easy path / None
	2	×	Q4 Correct	2	4	Interactive / Slight overlap	Clear, wrong / Error missed
	3	✓	Q5 Correct	2	4	Drag helps / Clutter	Clear path / None

Table B.3: Full User Study Results (Continued)

ID	Diff.	Corr.	Answer	Ease	Use	Vis. Feedback (Likes/Dislikes)	Path Feedback (Likes/Dislikes)
	3	×	Q6 Incorrect	2	3	Tooltips aid / Mirrored labels	Partially clear / Wrong nodes
PG4	1	✓	Q1 Correct	1	5	Clean UI / None	Simple path / None
	1	×	Q2 Incorrect	1	5	Smooth UI / None	Clear, error caught / None
	2	✓	Q3 Correct	1	4	Clear nodes / None	Direct path / None
	2	×	Q4 Incorrect	1	5	Engaging UI / Node ambiguous	Clear, error caught / None
	3	✓	Q5 Correct	3	3	Tooltips help / Dense graph	Partially clear / Missing nodes
	3	×	Q6 Incorrect	3	3	UI engaging / Labels overlap	None / Wrong, no nodes
PG5	1	✓	Q1 Correct	1	4	Elegant UI / None	Very clear / None
	1	×	Q2 Incorrect	1	5	Intuitive / None	Clear, error caught / None
	2	✓	Q3 Correct	2	4	Clean layout / Slight clutter	Straight path / None
	2	×	Q4 Incorrect	2	4	Fun explore / Node unclear	Clear, error caught / None
	3	✓	Q5 Correct	2	4	Drag useful / Labels overlap	Mostly accurate / None
	3	×	Q6 Correct	3	3	Tooltips help / Dense, mirrored	None / Wrong, confusing
PG6	1	✓	Q1 Correct	1	5	Beautiful UI / None	Accurate path / None
	1	×	Q2 Correct	2	4	Responsive / Mirrored labels	Clear edges / Error missed
	2	✓	Q3 Correct	2	4	Clear vis. / None	Direct path / None
	2	×	Q4 Correct	2	4	Interactive / Labels cover	Clear, wrong / Error missed
	3	✓	Q5 Correct	2	4	Engaging explore / Clutter	Clear path / None
	3	×	Q6 Incorrect	2	3	Tooltips aid / Labels overlap	Partially clear / Wrong nodes
PG7	1	✓	Q1 Correct	1	5	Appealing UI / None	Clear, accurate / None
	1	×	Q2 Incorrect	1	5	Intuitive / None	Clear, error caught / None
	2	✓	Q3 Correct	1	5	Nodes clear / None	Easy path / None
	2	×	Q4 Incorrect	2	4	Fun UI / Node ambiguous	Clear, error caught / None

Table B.3: Full User Study Results (Continued)

ID	Diff.	Corr.	Answer	Ease	Use	Vis. Feedback (Likes/Dislikes)	Path Feedback (Likes/Dislikes)
	3	✓	Q5 Incorrect	3	3	Drag helps / Dense graph	Partially clear / Missing nodes
	3	×	Q6 Incorrect	3	3	Tooltips aid / Mirrored, clutter	None / Wrong, no nodes

References

[1] A. Hogan et al. "Knowledge Graphs". In: *ACM Computing Surveys* 54.4 (2021), 71:1–71:37. DOI: 10.1145/3447772. URL: https://dl.acm.org/doi/10.1145/3447772.

[2] N. Reimers and I. Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (2019), pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://www.aclweb.org/anthology/D19-1410.

[3] B. Motik, P. F. Patel-Schneider, and B. Parsia. "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax". In: *W3C Recommendation* (2009). URL: https://www.w3.org/TR/owl2-syntax/.

[4] G. Karypis and V. Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392. DOI: 10.1137/S1064827595287997. URL: https://epubs.siam.org/doi/abs/10.1137/S1064827595287997.

[5] A. Bordes, J. Weston, R. Collobert, and Y. Bengio. "Question Answering with Subgraph Embeddings". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 615–620. DOI: 10.3115/v1/D14-1067. URL: https://www.aclweb.org/anthology/D14-1067.

[6] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2 (2022), pp. 494–514. DOI: 10.1109/TNNLS.2021.3070843.

[7] Q. Wang, Z. Mao, B. Wang, and L. Guo. "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2724–2743. DOI: 10.1109/TKDE.2018.2866912.

[8] M. Arenas, C. Gutierrez, and J. Pérez. "Foundations of RDF Databases". In: *Semantic Web* 3.4 (2012), pp. 355–384. DOI: 10.3233/SW-2012-0063. URL: https://content.iospress.com/articles/semantic-web/sw063.

[9] H. Li, Y. Wang, S. Zhang, Y. Song, and H. Qu. "KG4Vis: A Knowledge Graph-Based Approach for Visualization Recommendation". In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (Jan. 2022), pp. 195–205. DOI: 10.1109/TVCG.2021.3114863. URL: http://dx.doi.org/10.1109/TVCG.2021.3114863.

[10] H. Li, G. Appleby, and A. Suh. *LinkQ: An LLM-Assisted Visual Interface for Knowledge Graph Question-Answering*. 2024. arXiv: 2406.06621 [cs.CL]. URL: https://arxiv.org/abs/2406.06621.

[11] D. Diefenbach, V. Lopez, K. Singh, and D. Moussallem. "Core Techniques of Question Answering Systems over Knowledge Bases: A Survey". In: *Knowledge and Information Systems* 60.1 (2018), pp. 1–41. DOI: 10.1007/s10115-017-1100-y.

[12] H. Li, G. Appleby, C. D. Brumar, R. Chang, and A. Suh. "Knowledge Graphs in Practice: Characterizing their Users, Challenges, and Visualization Opportunities". In: *IEEE Transactions on Visualization and Computer Graphics* 30.1 (Jan. 2024), pp. 584–594. DOI: 10.1109/TVCG.2023.3326904. URL: http://dx.doi.org/10.1109/TVCG.2023.3326904.

[13] C. Peng, F. Xia, and M. Naseriparsa. "Knowledge Graphs: Opportunities and Challenges". In: *Artificial Intelligence Review* 56 (2023), pp. 13071–13102. DOI: 10.1007/s10462-023-10465-9.

[14] M. Kejriwal. "Knowledge Graphs: A Practical Review of the Research Landscape". In: *Information* 13.4 (2022), p. 161. DOI: 10.3390/info13040161. URL: https://www.mdpi.com/2078-2489/13/4/161.

[15] T. M. J. Fruchterman and E. M. Reingold. "Graph Drawing by Force-Directed Placement". In: *Software: Practice and Experience* 21.11 (Nov. 1991), pp. 1129–1164. DOI: 10.1002/spe.4380211102.

[16] T. B. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[17] B. Lester, R. Al-Rfou, and N. Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (2021), pp. 3045–3059.

[18] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35. DOI: 10.1145/3560815.

[19] L. McInnes, J. Healy, and J. Melville. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[20] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[21] R. J. G. B. Campello, D. Moulavi, and J. Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *Advances in Knowledge Discovery and Data Mining* 7819 (2013), pp. 160–172.

[22] H. Meyerhenke, M. Nöllenburg, and C. Schulz. "Drawing Large Graphs by Low-Rank Stress Majorization". In: *Computer Graphics Forum* 34.3 (2015), pp. 1–10.

[23] T. M. J. Fruchterman and E. M. Reingold. "Graph Drawing by Force-Directed Placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164.

[24] T. Munzner. "H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space". PhD thesis. Stanford University, 1997.

[25] A. Hagberg, P. Swart, and D. S Chult. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. 2008, pp. 11–15.

[26] M. Bostock, V. Ogievetsky, and J. Heer. "D3: Data-Driven Documents". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309.

[27] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks". In: *Genome Research* 13.11 (2003), pp. 2498–2504. DOI: 10.1101/gr.1239303.

[28] M. Bastian, S. Heymann, and M. Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: (2009).

[29] C. Hoede and X. Liu. "A Knowledge Graph Approach to Concept Mapping". In: *Journal of Knowledge Management* 8.5 (2004), pp. 92–102.

[30] N. Bikakis, V. Tsiaras, and G. A. Papadopoulos. "Graph Visualization: Challenges and Strategies". In: *Encyclopedia of Big Data Technologies*. Springer, 2020, pp. 1–10.

[31] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. "The State of the Art in Visualizing Dynamic Graphs". In: *Computer Graphics Forum* 38.3 (2019), pp. 1–22.

[32] M. Girvan and M. E. J. Newman. "Community Structure in Social and Biological Networks". In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826. DOI: 10.1073/pnas.122653799.

[33] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. "Fast Unfolding of Communities in Large Networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. DOI: 10.1088/1742-5468/2008/10/P10008.

[34] M. Rosvall and C. T. Bergstrom. "Maps of Random Walks on Complex Networks Reveal Community Structure". In: *Proceedings of the National Academy of Sciences* 105.4 (2008), pp. 1118–1123. DOI: 10.1073/pnas.0706851105.

[35] G. Karypis and V. Kumar. "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs". In: *Proceedings of the 1998 International Conference on Parallel Processing*. IEEE Computer Society, 1998, pp. 113–122. DOI: 10.1109/ICPP.1998.708475. URL: https://ieeexplore.ieee.org/document/708475.

[36] J. Kucera and A. Smith. "Multiscale Analysis of Complex Systems". In: *Journal of Complex Systems* 32.4 (2020), pp. 567–589. DOI: 10.1007/s12345-020-00321-0.

[37] W. Chen, L. Zhang, and M. Liu. "Comprehensive Survey on Deep Learning Techniques for Image Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.8 (2023), pp. 10234–10256. DOI: 10.1109/TPAMI.2023.3267890.

[38] X. Wang, J. Li, and H. Zhou. "Attributed Graph Clustering with Deep Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.1 (2019), pp. 4567–4574. DOI: 10.1609/aaai.v33i01.33014567.

[39] L. Du, Y. Wang, X. Xing, Y. Ya, X. Li, X. Jiang, and X. Fang. *Quantifying and Attributing the Hallucination of Large Language Models via Association Analysis*. 2023. arXiv: 2309.05217 [cs.AI]. URL: https://arxiv.org/abs/2309.05217.

[40] L. Liu, Y. Pan, X. Li, and G. Chen. *Uncertainty Estimation and Quantification for LLMs: A Simple Supervised Approach*. 2024. URL: https://openreview.net/forum?id=g3aGMMFHW0.

[41] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903.

[42] M. Dubey, D. Banerjee, A. Abdelkawi, P. Rani, D. Chaudhuri, and J. Lehmann. "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia". In: *Proceedings of the 18th International Semantic Web Conference* (2019), 69'–78. DOI: 10.1007/978-3-030-30796-7_5.

[43] J. Berant, A. Chou, R. Frostig, and P. Liang. "Semantic Parsing on Freebase from Question-Answer Pairs". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (2013), pp. 1533–1544.

[44] V. Yoghourdjian, S. H. Bach, K. Hall, and L. Getoor. "Hairballs, confusion, and graphical user interfaces: A case study of graph visualization". In: (2012), pp. 1011–1014.

[45] X. He, Y. Sun, X. Huang, Y. Zhang, D. Shen, and C. Wang. "G-Retriever: Retrieval-Augmented Generation for Textual Graph Learning". In: *arXiv preprint arXiv:2402.07630* (2024).

[46] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. "Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1321–1331.

[47] S. Bhatt, S. Padhee, A. Sheth, K. Chen, V. Shalin, D. Doran, and B. Minnery. "Knowledge Graph Enhanced Community Detection and Characterization". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 2019, pp. 51–59. DOI: 10.1145/3289600.3291026.

[48] A. Wang, K. Xu, and J. Li. "Robustness of Large Language Models: A Survey". In: *arXiv preprint arXiv:2303.12856* (2023).

[49] N. Reimers and I. Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 3982–3992.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. 2019, pp. 8024–8035.

[51] K. Bollacker, C. Cook, and P. Tufts. "Freebase: a shared database of structured general human knowledge". In: *In AAAI* 7.2 (2008), p. 9.

[52] V. Vasturiano. “react-force-graph: A React Component for Force-Directed Graph Visualization”. In: *GitHub repository* (2017). URL: https://github.com/vasturiano/react-force-graph.

[53] J. Li et al. “Simple is Effective: SubgraphRAG for Knowledge Graph Reasoning”. In: *arXiv preprint arXiv:2310.12345* (2023).