QirK: Question Answering via Intermediate Representation on Knowledge Graphs

Jan Luca Scheerer¹ Anton Lykov² Moe Kayali²
Ilias Fountalis³ Dan Olteanu⁴ Nikolaos Vasiloglou³
Dan Suciu²

¹ETH Zurich ²University of Washington ³RelationalAI, Inc. ⁴University of Zurich

Version as of: January 31, 2024

Abstract

We demonstrate QirK, a system for answering natural language questions on Knowledge Graphs (KG). QirK can answer structurally complex questions that are still beyond the reach of emerging Large Language Models (LLMs). It does so using a unique combination of database technology, LLMs, and semantic search over vector embeddings. The glue for these components is an intermediate representation (IR). The input question is mapped to IR using LLMs, which is then repaired into a valid relational database query with the aid of a semantic search on vector embeddings. This allows a practical synthesis of LLM capabilities and KG reliability.

A short video demonstrating QirK is available at https://youtu.be/6c81BLm0Z0U.

1 Answering Questions using Knowledge Graphs

Databases operate under the Closed World Assumption. Their domain is limited but they can answer complex queries correctly, completely, and efficiently. Large Language Models (LLMs) [5, 9] support an Open World Assumption: they can answer simple, open-ended queries from an impressively large number of domains. However, LLMs often return incorrect answers, tend to hallucinate, and fail to answer complex questions that are easily supported by databases. A recent systematic study of their coverage [8] found that "existing LLMs are still far from being perfect in terms of their grasp of factual knowledge, especially for facts of torso-to-tail entities."

We argue that Knowledge Graphs (KGs) offer the right middle ground for answering complex questions about a wide range of domains. Several large, open source KGs have been developed, e.g., Wikidata [10], Dbpedia [1], Yago [7]; see [11] for a recent survey. Open-source KGs are curated by the community, constantly updated and improved, and their coverage, while not as large as that of LLMs, is still impressive. KGs can naturally be stored in a database and queried with SPARQL or SQL. They can give accurate and complete answers to complex queries in many domains.

However, queries over KGs are very difficult to formulate. Wikidata has over 10,000 distinct properties and well over 100 million entities. *De facto* these constitute the *schema* of the database, and it is impossible, even for an expert, to be familiar with all of them. While KGs hold immense potential for a wide array of applications, their extensive and intricate schemas, however, pose a significant challenge as existing querying systems usually demand an intimate familiarity with the specific labels and structures in the graph.

We introduce QirK (\underline{Q} uestion Answering via \underline{I} ntermediate \underline{R} epresentation on \underline{K} nowledge Graphs), a system that uses a unique combination of LLMs, semantic search on vector embeddings, and database technology to answer complex questions on KGs. QirK addresses the previous challenge head-on, by allowing

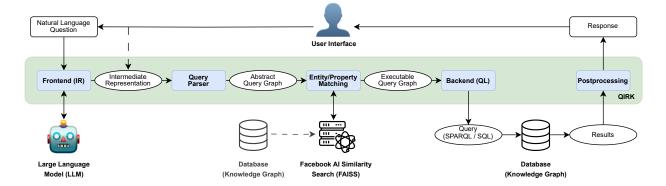


Figure 1: QirK Architecture. The natural language question is translated into an intermediate representation (IR) with the help of an LLM. Keywords in the IR are resolved to identifiers of semantically matching KG entities and properties to obtain an executable query graph, which is translated into SPARQL or SQL and evaluated using a database system hosting the KG.

users to formulate both natural language questions and loosely structured, intuitive queries with minimal natural language constraints. It uses an LLM to translate from natural language questions to an intermediate representation (IR), and a FAISS (Facebook AI Similarity Search) [3] index to resolve keywords in the IR to entities and properties in the KG. By combining the strengths of LLMs and databases, QirK paves the way for more effective and intuitive KG querying.

Recently, several systems have been developed that translate natural language to SQL. They often achieve over 90% accuracy on Spider [13], the standard benchmark for NLP to SQL translation. However, these systems, e.g., CatSQL [4], are primarily developed for relational databases with a given, and relatively small schema. Newer systems [12] have demonstrated the capability to directly generate SPARQL queries. This poses greater difficulty for LLMs and is less transparent to users and limited by its requirement to link keywords to a single entity/property. QirK overcomes this by using a much simpler intermediate representation that can be repaired flexibly into valid SPARQL and SQL queries.

2 QirK Architecture

Fig. 1 overviews the architecture of QirK. The user can pose natural language questions in QirK's user interface. Using an LLM¹, e.g., ChatGPT (https://chat.openai.com), Bard (https://bard.google.com), or Copilot (https://www.bing.com/chat), the question is translated into an intermediate representation (IR). The IR is a formal language, but still uses snippets of natural language to referr to entities, relationships between entities, properties, and qualifiers. QirK internally represents the IR as an abstract query graph, where relationships (relational atoms) become edges and entities (query constants and variables) become nodes.

The keywords in the IR are then matched against entities and properties available in the KG using an index storing high-dimensional vector embeddings of entities and properties derived from the KG. QirK requires that this index is pre-computed from the input KG before answering questions over it. Embeddings are extracted using the SentenceBERT model [6]. This matching step can yield several possible entities and properties that are in the KG and that are similar to the keywords in the IR. Each such matching comes with a similarity score (cosine similarity), which captures the confidence in the quality of the matching: a score value close to 1 (0) signals a good (poor) matching.

Using these matches, QirK assembles an executable query graph specialized to the vocabulary of the underlying KG. This graph is translated into a structured query language supported by the underlying database system: we currently translate to SPARQL, and, from there, to SQL (as our KG is stored in

¹The current implementation of QirK uses ChatGPT 3.5 [2].

PostgreSQL). Both SPARQL and SQL queries use correct identifiers for entities and properties in the input KG. They are not human readable, which makes a direct expression of such queries challenging for humans.

In the last step, the query is evaluated using a database system that hosts a relational encoding of the KG. The current implementation of QirK uses PostgreSQL. The query answers are presented to the user in a tabular form with additional confidence information. This confidence is the average score of matches of the keywords in the IR to actual entities and properties in the KG. As there may be several possible matches for the same keyword, each with its own score, there can be answers that are produced by different matches and therefore have different overall scores. QirK ranks the answers by inspecting the scores of the matched properties and entities.

Even though the obtained SPARQL/SQL query contains a list of potential entity/property identifiers for each keyword in the IR, it nevertheless remains selective. This is because most combinations of entities and properties from the lists associated with the different keywords in the query are not logically coherent and, thus, do not produce valid relationships in the KG. Therefore, QirK implicitly leverages the KG to identify semantically coherent combinations without requiring prior knowledge of meaningful associations.

3 **QirK** by Examples

Let us first consider the following question:

"Name people who have won both an Oscar for Merit and a Turing Award."

Current LLMs fail to answer this query (as of January 27, 2024): ChatGPT is unaware of any such individuals. Bard returns incorrect results (e.g., Steven Spielberg), while Copilot states it cannot find any such people. In contrast, QirK uses the Wikidata KG to provide the correct answer: Edwin Catmull.

QirK first translates the natural language question into an IR:

```
X: received_award(X, "Oscar for Merit");
  received_award(X, "Turing Award")
```

The IR syntax resembles first-order logic: X is a variable and we seek values for X such that the body of the query is satisfied, i.e., the relationship received_award holds between X and both "Oscar for Merit" and "Turing Award".

Evaluating a SPARQL or SQL encoding of this IR on Wikidata yields no answer, since neither the property received_award nor the entity Oscar for Merit exist in the KG. QirK addresses this by using the FAISS index to resolve predicates and entities in the KG that are semantically similar to those in the IR. To improve the accuracy of the matching, QirK takes the label, description, and the popularity into account. In this example, QirK resolves the predicate received_award to the intended property award received (P166). It also finds the entities Medal of Merit (Q7408872, score=0.70), Medal for Merit to Culture (Q1702885, score=0.74), Medal for Merit (Q1307005, score=0.68), and Gold Medal of Merit in the Fine Arts (Q3753203, score=0.73) as well as the intended entity Academy Award of Merit (Q8624, score=0.77) as potential candidates for the keyword "Oscar for Merit". For the keyword "Turing Award", QirK retrieves the matching entities: Turing machine (Q163310, score=0.65), Category:Turing Award (Q9241105, score = 0.67), Turing (Q490481, score = 0.67), Alan Turing (Q7251, score=0.70), and the intended entity Turing Award (Q185667, score=0.89).

QirK rewrites the IR into an executable query graph, where each keyword is replaced by the list of the retrieved identifiers. It then compiles the query graph into SPARQL or SQL and evaluates it using a relational database system. We next show the generated SPARQL query (the SQL query is similar, albeit more verbose):

```
SELECT ?A1 ?C3 ?A2 ?HO WHERE {
    ?HO ?C3 ?A1. ?HO ?C3 ?A2.
    FILTER (?A1 IN (wd:Q7408872, wd:Q8624, wd:Q1702885,
```

```
wd:Q1307005, wd:Q3753203)
&& ?C3 IN (wdt:P166)
&& ?A2 IN (wd:Q163310, wd:Q185667, wd:Q490481,
wd:Q9241105, wd:Q7251)) }
```

The above SPARQL query has the following variables: ?HO iterates over individuals, ?A1 and ?A2 iterate over the possible entities matched to Oscar for Merit and Turing Award respectively, and ?C3 is bound to the property award received with identifier P166.

Even though the generated SPARQL query specifies a list of potential matches for each of the two entities ?A1 and ?A2, the conjunction of their relationships with the award received property is only satisfied by logically coherent combinations, namely ?A1 = wd:Q8624 and ?A2 = wd:Q185667. Without QirK one would have to write the following SPARQL query:

```
SELECT ?HO WHERE {
    ?HO wdt:P166 wd:Q8624. ?HO wdt:P166 wd:Q185667.}
```

Evidently, this query is significantly harder to express than the natural language question given as input to QirK.

After the query is evaluated by a database system and the answers are retrieved by QirK, QirK re-ranks the answers based on their similarity scores. This is done by analyzing the concrete assignment to ?A1, ?C3 and ?A2 for each answer and averaging their similarity scores. These results are then presented to the user.

Structurally complex questions. QirK can also answer structurally more complex questions, such as:

"List movies where the director is married to a member of the cast."

Figure 2 (left) shows an excerpt of QirK's answers to this question. QirK finds 3324 movies that satisfy the intricate triangle relationships between movies, directors, and their spouse actors.

Large Language Models typically provide a small set of answers to such questions. For instance, ChatGPT only gives four (correct) answers. Bard gives 20 answers, several of them are partially incorrect as they do not satisfy the three binary relationships. Such answers correctly identify actors and cast members involved in the same movie, yet without being married to each other, e.g.: (i) 1990: Pretty Woman (Director: Garry Marshall, Cast: Julia Roberts); (ii) 1994: Pulp Fiction (Director: Quentin Tarantino, Cast: Uma Thurman); (iii) 2001: The Lord of the Rings: The Fellowship of the Ring (Director: Peter Jackson, Cast: Liv Tyler). Bing Chat is able to retrieve a website² with concrete examples and therefore provides a limited number (ten) of correct results. A clear difficulty of using LLMs for such questions is their unreliability and the time-consuming effort to check whether their answers are correct. In contrast, QirK observes all relationships stated in the question and can only provide answers that satisfy all these relationships.

QirK expresses this question in IR as the following cyclic query:

```
X: movie(X); director(X,Y); married(Y,Z); cast(X,Z)
```

Using the vector index, QirK retrieves the following semantically similar properties (and identifiers) for the keywords in the IR. For the keyword movie, it finds: Movie (Q2512663) (disambiguation page), film (Q11424), Film (Q12362625) (album), B movie (Q223770) (genre), and Movie Movie (Q1405677) (film by Stanley Donen). The keywords director, married, and cast are mapped to the properties director (P57), spouse (P26), and cast member (P161) respectively. QirK combines these matches with the IR to generate the SPARQL query (also depicted in Fig. 2):

²https://rb.gy/zyg6lx

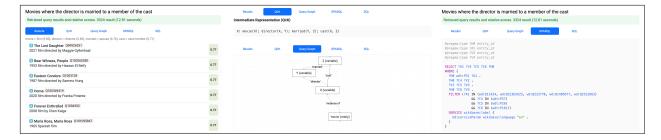


Figure 2: Snapshot of QirK's user interface. (left) Users can write questions in natural language or in QirK's intermediate representation. QirK's answer is presented in tabular form. (middle) The query graph inferred by QirK from the natural language question is shown for inspection. (right) QirK displays the SPARQL and SQL queries generated from the executable query graph. The queries include identifiers of entities/properties from the underlying KG obtained using the FAISS index.

Datatypes. With the exception of globe coordinates, QirK supports all Wikidata datatypes: entity_id, string, date, numeric, and qualifiers. QirK resolves the datatypes automatically, where possible, or uses those supplied in the IR. The type of a query variable must be consistent across the query, but need only be declared for one of its occurrences. For instance, the question: "When was Alan Turing born?" is transformed into the following IR:

```
X: date_of_birth("Alan Turing", X / date)
```

We can also ask for the US presidents and their heights:

```
X, Y: position(X, "President of the United States");
    height(X, Y / numeric)
```

where we explicitly state that the variable ${\tt Y}$ is of type ${\tt numeric}.$

Qualifiers. QirK supports qualifiers in its intermediate representation. For instance, the question "When did Barack Obama become president?" can be expressed as follows in QirK's IR:

```
Y: X := holds_position("Barack Obama", "President");
start_time(X, Y / date)
```

The qualifier X is assigned to the fact holds_position("Barack Obama", "President") and used to make a further statement. This statement is expressed by the second relational atom in the IR, where we ask for the start_time of the previous fact.

Aggregation. QirK also supports min/max aggregation. For instance, we can ask for the height of the tallest US president:

```
MAX(Y): position(X,"President of the United States");
    height(X,Y / numeric)
```

QirK's functionality can be extended to support further aggregates such as count, sum, and average.

4 User Interaction

QirK provides a web user interface. Fig. 2 depicts three snapshots of QirK answering the movie question from Sec. 3. QirK's interface has one tab to depict each transformation step of the user question, from the initial statement in natural language to the intermediate representation, and finally to SPARQL and SQL queries.

On the Search tab (not shown), the user can enter the question either in natural language or in QirK's intermediate representation.

On the *Results* tab, the user can inspect the answers. QirK groups the answers by the mapping from the keywords in the question to KG entities and properties. All answers in a group have the same confidence score. Both the mapping and the associated score are presented for each answer group. Each answer comes with an entity identifier and a link to the Wikidata page describing it.

The Intermediate Representation tab graphically depicts the IR query graph made up of the query variables and relations.

The SPARQL and SQL tabs show the final structured queries. Fig. 2 (right) shows the SPARQL query for the movie question.

5 Demonstration Scenarios

The demonstration will show how QirK answers a combination of ad-hoc and previously prepared questions designed to highlight specific strengths and weaknesses of QirK:

Reliability. QirK can provide high-quality answers to questions where LLMs fail (hallucinate or are not capable of providing an answer). Examples of such questions include asking for people or facts that satisfy a conjunction or disjunction of properties, such as asking for people who won as Oscar and a Turing award (Sec. 3).

Interpretability. QirK allows users to scrutinize each stage of the retrieval process. Through the transparent presentation of the explicit mapping of keywords to entities/properties in the KG, users can assess the accuracy of results. This stands in stark contrast to LLMs, which offer no visibility into the retrieval stages.

Robustness. QirK is robust to minor changes to the question that preserve its meaning such as replacing keywords with synonyms and accommodating typos, e.g.,: variations such as holds_position instead of position, actor instead of cast, or Oscar for Merit instead of Academy Award for Merit.

Expressiveness. Using aggregates, types, and qualifiers, QirK can provide answers to complex queries, as highlighted in Sec. 3.

Limitations. QirK may not yet offer satisfactory answers to questions that are ambiguous or require extensive reasoning, e.g., "Does it make sense to bring Euro on a trip to the US?".

References

[1] S. Auer and et al. Dbpedia: A nucleus for a web of open data. In *ISWC*, volume 4825, pages 722–735, 2007.

- [2] T. B. Brown and et al. Language models are few-shot learners. CoRR, abs/2005.14165, 2020.
- [3] M. Douze and et al. The faiss library. 2024.
- [4] H. Fu and et al. Catsql: Towards real world natural language to sql applications. *Proc. VLDB Endow.*, 16(6):1534–1547, 2023.
- [5] OpenAI. GPT-4 technical report. CoRR, abs/2303.08774, 2023.
- [6] 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. Association for Computational Linguistics, 11 2019.
- [7] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In WWW, pages 697–706, 2007.
- [8] K. Sun, Y. E. Xu, H. Zha, Y. Liu, and X. L. Dong. Head-to-tail: How knowledgeable are large language models (llm)? A.K.A. will llms replace knowledge graphs? *CoRR*, abs/2308.10168, 2023.
- [9] H. Touvron and et al. Llama: Open and efficient foundation language models. CoRR, abs/2302.13971, 2023.
- [10] D. Vrandecic and M. Krötzsch. Wikidata: a free collaborative knowledgebase. Commun. ACM, 57(10):78–85, 2014.
- [11] G. Weikum, X. L. Dong, S. Razniewski, and F. M. Suchanek. Machine knowledge: Creation and curation of comprehensive knowledge bases. *Found. Trends Databases*, 10(2-4):108–490, 2021.
- [12] S. Xu and et al. Fine-tuned llms know more, hallucinate less with few-shot sequence-to-sequence semantic parsing over wikidata, 2023.
- [13] T. Yu and et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, pages 3911–3921, 2018.