

QUINT: Interpretable Question Answering over Knowledge Bases

Abdalghani Abujabal¹, Rishiraj Saha Roy¹, Mohamed Yahya² and Gerhard Weikum¹

¹Max Planck Institute for Informatics, Saarland Informatics Campus, Germany

{abujabal, rishiraj, weikum}@mpi-inf.mpg.de

²Bloomberg L.P., London, United Kingdom

myahya6@bloomberg.net

Abstract

We present QUINT, a live system for question answering over knowledge bases. QUINT automatically learns role-aligned utterance-query templates from user questions paired with their answers. When QUINT answers a question, it visualizes the complete derivation sequence from the natural language utterance to the final answer. The derivation provides an explanation of how the syntactic structure of the question was used to derive the structure of a SPARQL query, and how the phrases in the question were used to instantiate different parts of the query. When an answer seems unsatisfactory, the derivation provides valuable insights towards reformulating the question.

1 Introduction

Motivation. A KB-QA system takes a natural language utterance as input and produces one or more crisp answers as output (Bast and Haussmann, 2015; Berant et al., 2013; Reddy et al., 2014; Yih et al., 2015). This is usually done through semantic parsing: translating the utterance to a formal query in a language such as SPARQL, and executing this query over a KB like Freebase (Bollacker et al., 2008) or YAGO (Suchanek et al., 2007) to return one or more answer entities.

In addition to answering questions, a KB-QA system should ideally be able to explain how an answer was derived i.e., how the system understood the users’ questions. While rapid progress is being made on the KB-QA task, the quality of answers obtained from KB-QA systems are far from

perfect. This is due to a combination of factors related to the ambiguity of natural language, the underlying data (e.g., KB incompleteness, gaps in lexicon coverage) and the KB-QA systems themselves (e.g., errors in named entity recognition and disambiguation, query ranking). Explanations help address this gap in two ways: (i) helping users gain confidence when correct answers are returned, and (ii) making sense of the limitations of the system by looking at explanations for wrong answers, possibly providing cues to work around them. For an expert user, explanations also contribute to *traceability*: identifying the exact point of failure in the KB-QA system pipeline, which can be used for subsequent debugging.

In this work, we demonstrate QUINT (Abujabal et al., 2017), a state-of-the-art KB-QA system that gives step-by-step explanations of how it derives answers for questions. Furthermore, when QUINT is unable to link a specific phrase in the question to a KB item, it asks the user to reformulate the phrase. Such reformulations can be used to improve various components in the KB-QA pipeline such as underlying lexicons. QUINT takes the first step towards enabling interactive QA in the future, where the system can ask the user about parts of the question that it is unsure about.

Example. Take the question “Where was Martin Luther raised?”: QUINT returns `Eisleben` in Germany as the top answer. A quick look by the user at the derivation reveals that (i) ‘Martin Luther’ was mapped to the KB entity `MartinLuther`, the theologian, and (ii) ‘raised’ was interpreted as the KB predicate `placeOfBirth`. For (i), if the user had intended the US activist `MartinLutherKing` instead, a simple reformulation with “martin luther king” in the input returns `Atlanta`, the US city where Luther King was born. On the other hand, for (ii), if the birthplace was not the specific intent, a quick

²Work done while at the Max Planck Institute for Informatics

rephrasing of the question to “Where did Martin Luther live?” results in Saxony-Anhalt, which is derived from the predicate `placesLived`.

Motivated by the need for interpretable question answering, QUINT’s approach to KB-QA relies on *role-aligned templates*, where each template consists of an *utterance template* based on a dependency parse pattern and a corresponding *query template* based on the SPARQL query language. The template (i) specifies how to chunk an utterance into phrases, (ii) guides how these phrases map to KB primitives by specifying their semantic roles as predicates, entities, or types, and (iii) aligns syntactic structure in the utterance to the semantic predicate-argument structure of the query.

Limitations of past work. Prior template-based approaches rely on a set of *manually* defined rules or templates to handle user questions (Barrant et al., 2013; Fader et al., 2013, 2014; Unger et al., 2012; Yahya et al., 2013; Yao and Durme, 2014; Zou et al., 2014). The main drawback of these approaches is the limited coverage of templates, making them brittle when it comes to unconventional question formulations. In contrast, QUINT automatically learns templates from question-answer pairs.

Embedding-based methods (Bordes et al., 2014; Dong et al., 2015; Yang et al., 2014; Xu et al., 2016) map questions, KB entities, and subgraphs to a shared space for KB-QA without explicitly generating a semantic representation. This makes it difficult for such systems to generate fine-grained explanations to users.

Other approaches to KB-QA (Bast and Haussmann, 2015; Yih et al., 2015) over-generate query candidates for a given utterance with no fine-grained alignments to map natural language phrases in a question onto different KB items, making explainability challenging.

Contribution. The key contribution of this demo paper is a live online KB-QA system that visualizes the derivation steps for generating an answer, and thus takes the first steps towards explainable question-answering. The demo is available at the following URL: <https://gate.d5.mpi-inf.mpg.de/quint/quint>.

2 QUINT

We now give a brief overview of QUINT (Abujabal et al., 2017), the KB-QA system driving our demonstration. QUINT has a training phase

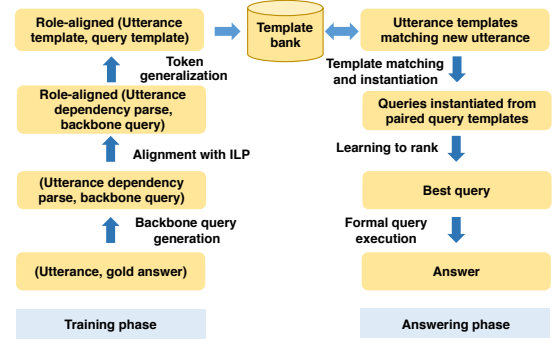


Figure 1: Overview of QUINT.

for automatically learning templates and a query ranking function, and an answering phase where templates are used to instantiate queries that are ranked by the learned function. Figure 1 shows a block diagram for QUINT.

2.1 Training phase

The input to QUINT’s training phase is a set of natural language utterances $u \in U$ and the corresponding gold answer set A_u from a KB such as the one shown in Figure 2. An example of a training utterance is $u =$ “Where was Obama educated?”, which is paired with the answer set $A_u = \{\text{ColumbiaUniversity}, \text{HarvardUniversity}, \text{PunahouSchool}\}$. First, entity mentions in each utterance u are detected and disambiguated to Freebase entities using the AIDA system (Hoffart et al., 2011).

Next, QUINT heuristically constructs a query to capture the question, the guiding intuition being that the correct query connects entities in a question u to an answer entity $a \in A_u$. To do this, QUINT starts by finding, for each answer entity $a \in A_u$, the smallest subgraph in the KB that contains all the entities detected in u and a (black nodes in Fig. 2 for $a = \text{ColumbiaUniversity}$). This subgraph is then extended by augmenting it with all *type nodes* connected to a (gray nodes `Organization` and `EducationalInstitution` in Fig. 2). This subgraph is transformed into a *backbone query* \hat{q} by replacing the answer node (a) and any `cvt` nodes with distinct variables (`cvt` nodes are used to represent n -ary relations in Freebase). The resulting \hat{q} is shown in Figure 3. This is followed by aligning the components of \hat{q} and the dependency parse of u . The alignment problem is formulated as a constrained optimization and the best alignment m is obtained using Integer Linear Programming (ILP).

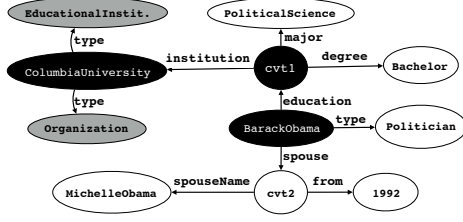


Figure 2: An example KB fragment.

To connect natural language phrases in a question to KB items, QUINT uses two kinds of weighted lexicons, a predicate lexicon \mathcal{L}_P and a type lexicon \mathcal{L}_C . Entities, as mentioned above, are dealt with using an off-the-shelf named entity recognition and disambiguation system. The output of the ILP solver tells us which tokens in u are used to instantiate which KB items in the backbone query \hat{q} . Nodes in the dependency parse of u as well as nodes and edges in \hat{q} that are not part of the alignment m are removed from the dependency parse of u and \hat{q} , respectively. In our running example, this results in dropping the node `EducationalInstitution` from \hat{q} (Fig. 3). The obtained alignment is then generalized by dropping concrete values in both u and \hat{q} , which are referred to as an utterance template u_t and a query template q_t , respectively. This role-aligned template pair of (u_t, q_t) is added to a template repository T . The process is repeated for each training instance (u, A_u) . However, since several utterances are likely to have similar syntactic structure, the number of templates $|T| \ll |U|$. Figure 4 shows the generated template from this instance.

Finally, as part of the training phase, QUINT trains a ranking function to rank the queries generated from matching a question against the template repository T . A learning-to-rank framework with a random forest classifier (Bast and Haussmann, 2015) is used to model a preference function for a pair of queries, given an input utterance.

2.2 Answering phase

When the trained system receives a new utterance u' from the user, the dependency parse of u' is matched against the utterance templates in T . For every match, the paired query template is instantiated using the alignment information together with the underlying lexicons. Thus, a set of candidate queries are obtained which are then ranked using the learning-to-rank framework. Finally, the answer of the top-ranked query is shown to the user.

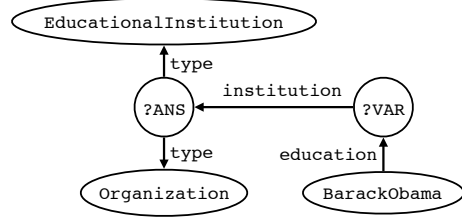


Figure 3: Backbone query \hat{q} generated from the utterance “Where was Obama educated?” and the answer entity `ColumbiaUniversity`.

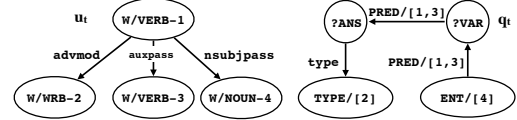


Figure 4: A pair of (utterance, query) templates. Shared numbers indicate alignment. W is a placeholder for any word. $ENT/PRED/TYPE$ are the semantic roles of nodes and edges.

3 Demonstration

We now give a walkthrough of our demonstration which shows QUINT’s ability to explain its answers to both normal and technically proficient users through brief and detailed explanations, respectively. We use the question “Where was Martin Luther raised?” to drive this section.

3.1 Question and Answers

Figure 5 shows the main window of QUINT where the starting point for a user is the question box or one of the sample questions provided. An expert user can make several configuration choices (Fig. 5, bottom right):

- Model to load: a model includes a set of templates learned offline, and a corresponding learned query ranking function. The choices correspond to the training part of the WebQuestions (Berant et al., 2013) and Free917 (Cai and Yates, 2013) datasets.
- Whether to add answer type to queries: most KB-QA systems do not capture fine-grained types, while QUINT does so by design.
- The number of top-ranked queries to show answers for.
- The number of decision trees for the learning-to-rank module: this is used for query ranking during the answering phase (Sec. 2.2).

The answers to each of the top- k ranked queries

QUINT: Interpretable Question Answering Over Knowledge Bases

Where was Martin Luther raised? Answer

Ask a question! OR Choose a question!

You can choose:

1. Data resource QUINT was trained on
2. Mode of operation
3. Number of SPARQL queries
4. Number of decision trees used by the Learning-to-Rank module

Sample Questions

- who played any squirrel on bad teacher?
- where did barack obama go to college?
- what is the currency of germany?
- birthplace of einstein?
- what governmental system does the us have?

Advanced Options

Training Resource: ☒ WebQuestions ☐ Free#17

QUINT Mode: ☒ Typed ☐ Untyped

Top-K Queries: ☐ 1 ☐ 3 ☒ 5

Number of Decision Trees for LTR: ☐ 90 ☐ 70 ☒ 40

Figure 5: User input.

Rank 1 Query Rank 2 Query Rank 3 Query Rank 4 Query Rank 5 Query

Brief Explanation >

Detailed Explanation >

Answer(s)

Eisleben 🌐

Eisleben is a town in Saxony-Anhalt, Germany. It is famous as the hometown of Martin Luther, hence its official name is Lutherstadt Eisleben. As of 2005, Eisleben had a population of 24,552. It lies on the Halle-Kassel railway. Eisleben is divided into old and new towns; the latter of which was created for Eisleben's miners in the 14th century. Eisleben was the capital of the district of Mansfelder Land and is the seat of the Verwaltungsgemeinschaft Lutherstadt Eisleben.




Figure 6: Retrieved answers from the KB.

in response to a question are given in individual tabs, as shown in Figure 6. To give more context to the user, wherever applicable, each answer is accompanied by a picture, a Wikipedia link and a short description from Freebase. For our running example question, the answer is `Eisleben`, where Martin Luther was born, for the best query. If we explore the rank-2 query, the answer is `Saxony-Anhalt`, which is the province in Germany where he lived.

3.2 Explanations

Explanations show how the given answers were derived. The ability to generate and display explanations is the core contribution of this system demonstration. QUINT generates two types of explanations: (i) a brief explanation geared towards non-technical users and (ii) a detailed explanation geared towards more advanced users.

Brief explanations provide an accessible and quick way for users to see how QUINT understood various parts of the question, resulting in the given set of answers. Figure 7 shows such an explanation for our running example. Brief explanations

Brief Explanation ▼

QUINT understood your question as follows:

- The phrase “martin luther” is interpreted as Martin Luther
- The words “was, raised” are interpreted as the relation Place of birth

Figure 7: QUINT’s brief explanation.

are particularly useful for normal users who are interested in confirming whether a given answer comes from interpreting the question as they intended. Where this is not the case, such an explanation guides users in reformulating their question to allow QUINT to better understand it.

Detailed explanations are geared towards advanced users who are familiar with dependency parses and SPARQL queries. A detailed explanation shows the derivation steps which roughly correspond to the right hand side of the diagram in Figure 1. First, the dependency parse of the input question is shown (Fig. 8). Below that is the *matching template* consisting of an utterance template that fits the utterance, and the corresponding query template that will be instantiated to generate a query. Shared numbers between the utterance template and the query template indicate alignment, i.e., which tokens in the utterance are used to instantiate which KB items in the query. In this example, we can see that the verb phrase ‘*was raised*’ and the noun phrase ‘*Martin Luther*’ are used to instantiate the KB-predicate and the KB-entity, respectively.

For a user to understand why QUINT maps a certain syntactic structure to a certain semantic form in a template, the user can view some training instances that produced the template. This is achieved by clicking *Template Provenance* (Fig. 9). For our example, two such instances are the questions “*Where was Obama educated?*” and “*Where are Riddell Helmets manufactured?*”.

Finally, the user is shown the SPARQL query that was executed to retrieve the shown answers (Fig. 10). The node labeled ?ANS is the target answer node, and nodes marked as ?VAR are intermediate join variables. Additionally, an alignment table with KB items is shown based on the alignments induced from the template (lower part of Fig. 10). For KB items, we find that ‘*Martin Luther*’ was mapped to the theologian `MartinLuther` (other possibilities include the German diplomat with the same name

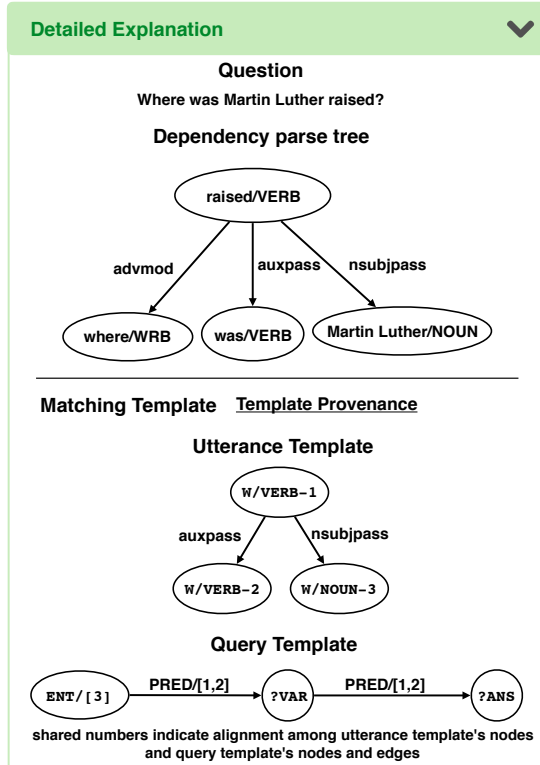


Figure 8: Question dependency parse and matched utterance and query template.

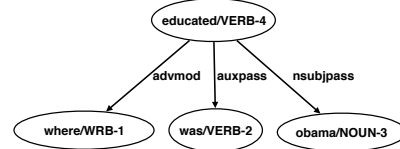
and MartinLutherKing), and the lemmatized phrase ‘be raise’ was mapped to the KB-predicate `placeOfBirth`. Looking at the rank-2 query, we find that ‘raise’ was interpreted as `placesLived` in the KB. This is an alternative intention when the slightly ambiguous phrase ‘raised’ is used.

4 Use Cases

When QUINT returns the correct answer, explanations allow the user to confirm its correctness by seeing how it was derived. When, on the other hand, things go wrong and an answer seems incorrect, explanations can give the user a better understanding of the limitations of the system (e.g., data incompleteness), and insights into overcoming these limitations (e.g., question rephrasing). We discuss some of these possible scenarios below, accompanied by real examples.

Incomplete lexicon. QUINT uses different lexicons to map the phrases in an utterance onto KB items (Sec. 2). These lexicons are inherently incomplete. For example, QUINT could not correctly answer the utterance “countries of euro-union” (implying the European Union) since the phrase ‘euro-union’ does not have an entry in QUINT’s lexicons. Therefore it shows the fol-

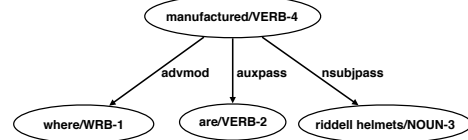
Training Utterance: where was obama educated?



Training Query



Training Utterance: where are riddell helmets manufactured?



Training Query

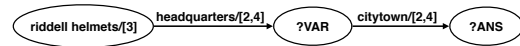
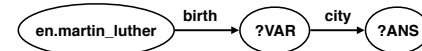


Figure 9: Structurally similar training instances to “Where was Martin Luther raised?”.

Query



Alignment

Lemmatized Phrase	Semantic Item	Type of Semantic Item
“Martin Luther”	en.martin_luther	Entity
“raise be”	birth, city	Predicate

Figure 10: SPARQL KB-query and KB items.

lowing message urging the user to reformulate the phrase: *The phrase euro-union in your input question could not be interpreted. Please reformulate the phrase.*

Incorrect query ranking. QUINT uses a learning-to-rank framework for ranking SPARQL queries. Sometimes, the most appropriate query is not ranked at the top. For example, for the utterance “What is the former currency of Germany?”, the top answer is Euro, which is incorrect. This is because the alignment information in the matching template fails to capture ‘former’. However, if the user explores the top-5 SPARQL queries, she finds that a query with the correct predicate `currencyFormerlyUsed` (as opposed to the incorrect `currencyUsed`) is indeed there and retrieves the desired answers `DeutscheMark`, `GermanRentenmark` and `GermanPapiermark`.

Incorrect disambiguation to KB items. Sometimes, the phrases in the input utterance are linked to wrong KB items as originally intended by the user. For example, for the utterance “What

language group does English belong to?”, ‘English’ gets mapped to the wrong entity `England`, resulting in an unexpected answer. When the user sees the brief explanation (Fig. 7), she can immediately observe this incorrect mapping. Subsequently, she may rephrase it as ‘English language’ which may produce the desired answer.

Missed answer type constraints. Answer typing plays an important role in ensuring precise QA. For example, in “Which college did Michelle Obama go to?”, the user explicitly specifies that she is looking for colleges, as opposed to “Where did Michelle Obama study?”, which, for example, could include her high school as well. Here ‘college’ is mapped to the KB type `University` and only when this constraint is added to the SPARQL query do we get the desired answer set `HarvardLawSchool` and `PrincetonUniversity`.

No matching templates. Sometimes an utterance is syntactically out of scope for QUINT: it has never seen similar instances during training. For example: “Germany’s capital during the forties?”. In such cases, QUINT raises the message *We could not find any matching templates. Please reformulate your question.*

5 Conclusion

We presented a demonstration of QUINT, a system that uses automatically learned templates to provide interpretable answers to natural language questions over knowledge bases. When a user gets an answer to her question, our demonstration allows her to view the details of how the answer was derived. This improves her confidence in case of correct answers, while giving her a better understanding of the limitations of the QA system in case of mistakes, and how to work around them.

References

- Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated template generation for question answering over knowledge graphs. In *WWW*.
- Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *CIKM*.
- Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*.
- Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*.
- Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *EMNLP*.
- Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*.
- Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *ACL*.
- Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *ACL*.
- Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open Question Answering over Curated and Extracted Knowledge Bases. In *KDD*.
- Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenu, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *EMNLP*.
- Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *TACL*.
- Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *WWW*.
- Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In *WWW*.
- Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *ACL*.
- Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. 2013. Robust question answering over the web of linked data. In *CIKM*.
- Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. 2014. Joint relational embeddings for knowledge-based question answering. In *EMNLP*.
- Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with freebase. In *ACL*.
- Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.
- Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*.