# Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs

Gaurav Maheshwari[1,3⋆], Priyansh Trivedi[1,3⋆], Denis Lukovnikov[1⋆], Nilesh Chakraborty[1⋆], Asja Fischer[2], and Jens Lehmann[1,3]

[1] Smart Data Analytics (SDA) Group, Bonn, Germany
[2] Ruhr-University, Bochum, Germany
[3] Fraunhofer IAIS, Dresden, Germany

**Abstract.** In this paper, we conduct an empirical investigation of neural query graph ranking approaches for the task of complex question answering over knowledge graphs. We propose a novel self-attention based *slot matching* model which exploits the inherent structure of query graphs, our logical form of choice. Our proposed model generally outperforms other ranking models on two QA datasets over the DBpedia knowledge graph, evaluated in different settings. We also show that domain adaption and pre-trained language model based transfer learning yield improvements, effectively offsetting the general lack of training data.

## 1 Introduction

Knowledge graph question answering (KGQA), where natural language questions like "What is the population of the capital of Germany?" can be answered by lookup and composition of one or many facts from a knowledge graph (KG), has garnered significant interest in the Natural Language Processing (NLP) and Semantic Web community.

Numerous approaches [7, 2, 22] use semantic parsing to create an *ungrounded* expression of a given natural language question (NLQ), and then ground it w.r.t. a target KG. Here, *grounding* refers to linking elements in the expression with elements (i.e. entities and predicates) in the KG. While this approach suits the non-trivial task of handling wide syntactic and semantic variations of a question during parsing, it needs to handle lexical as well as structural mismatch between the generated expression and the target KG during grounding. For instance, the predicate $mother(x_a, x_b)$, parsed from a question, might be represented as $parent(x_a, x_b) \wedge gender(x_a, female)$ in a KG. Failing to anticipate and tackle these issues can lead to undesirable situations where the system generates expressions which are illegal w.r.t. the given KG.

We focus on an alternate family of approaches [24, 23] which, given an NLQ, first generate a list of formal expressions describing possible candidate queries which are in accordance with the KG structure, and then rank them w.r.t. the NLQ. We use a custom grammar called *query graph* to represent these candidate expressions, comprised of paths in the KG along with some auxiliary constraints. In this work, we propose a novel *slot matching* model for ranking query graphs. The proposed model exploits the structure of query graphs by using attention to compute different representations of the

⋆ These four authors contributed equally

question for each predicate in the query graph. We compare our models against several baseline models by evaluating them over two DBpedia [12] based KGQA datasets namely, LC-QuAD [20] and QALD-7 [21].

Furthermore, we appropriate bidirectional transformers (BERT) [6] to be used in the slot matching configuration, thereby enabling the use of large-scale pre-trained language models for the task. To the best of our knowledge, this is the first work that explores their use for KGQA.

Finally, we also investigate the potential of transfer learning in KGQA by fine-tuning models trained on LC-QuAD on the much smaller QALD-7 dataset, resulting in a significant improvement in performance on the latter. We thereby demonstrate the efficacy of simple transfer learning techniques for improving performance of KGQA on target domains that lack training data.

The major contributions of this work are summarized as follows:

- A novel ranking model which exploits the structure of query graphs, and uses multiple attention scores to explicitly compare each predicate in a query graph with the natural language question.
- An investigation of fine-tuning based transfer learning across datasets and the use of pre-trained language models (BERT [6]) for the KGQA task.

Our experiments show that the proposed slot-matching model outperforms the baseline models, and that it can be combined with transfer learning techniques to offset the lack of training data in target domain. We have made the source code of our system, and the experiments publicly available at `https://github.com/AskNowQA/KrantikariQA`.

## 2 Related Work

In this section we briefly summarize existing approaches for KGQA and transfer learning techniques relevant to the use of pretrained language models for downstream tasks.

### 2.1 KG Question Answering

Traditional semantic parsing based KGQA approaches [2, 5, 7, 8, 19, 22] aim to learn semantic parsers that generate *ungrounded* logical form expressions from NLQs, and subsequently ground the expressions semantically by querying the KG.

In recent years, several papers have taken an alternate approach to semantic parsing by treating KGQA as a problem of semantic graph generation and ranking of different candidate graphs. [1] compare a set of manually defined query templates against the NLQ and generate a set of grounded query graph candidates by enriching the templates with potential predicates. Notably, [23] create grounded *query graph* candidates using a staged heuristic search algorithm, and employ a neural ranking model for scoring and finding the optimal semantic graph. The approach we propose in this work is closely related to this. [24] use a hierarchical representation of KG predicates in their neural query graph ranking model. They compare their results against a local sub-sequence alignment model with cross-attention [16] (originally proposed for the natural language

inferencing task [3]). We adapt the models proposed by both [16] and [24] to our task, and compare them against the ranking model we propose (See Sec. 4.2).

### 2.2 Transfer Learning from pre-trained Language Models

Recently, multiple approaches have been proposed exploiting transfer learning from pre-trained language models for downstream NLP tasks. They typically use a (downstream) task-agnostic architecture that undergoes time-consuming pre-training over large-scale general-domain corpus, and then is fine-tuned for different target tasks separately. [10] propose an innovative mechanism of fine-tuning long-short-term-memory (LSTMs) [9] and a wide repertoire of regularization techniques which prevent overfitting and catastrophic forgetting. The use of transformers in a similar transfer learning setting was proposed by [18]. The architecture was augmented by [6] enabling bidirectional training of transformers by masking random tokens in the input sequence, and training the model to predict those missing words. Generally, these approaches have achieved state-of-the-art results over multiple NLP tasks, including text classification, reading comprehension, named entity recognition. Motivated by their success, we investigate the effect of leveraging these transfer learning approaches for KGQA (as described in Sec. 5.3). Parallel to our work, [14] also explored the use of pre-trained transformer for the task of simple question answering over KG. They showed that with a fraction of training data, using pre-trained language models can achieve almost similar performance to those trained from scratch.

## 3 Background

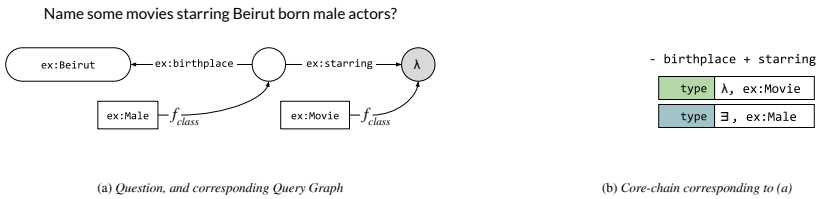
In this section we will give a formal definition of the task of KGQA and a description of the employed query graph language.

### 3.1 Problem Formulation

Let $K \subseteq (\mathcal{E} \times \mathcal{P} \times (\mathcal{E} \cup \mathcal{L}))$ be a KG where $\mathcal{E} = \{e_1 \dots e_{n_e}\}$ is the set of entities, $\mathcal{L}$ is the set of all literal values, and $\mathcal{P} = \{p_1 \dots p_{n_p}\}$ is the set of predicates connecting two entities, or an entity with a literal.

Given $K$, and a natural language question $Q$, the KGQA task can be defined as generating an expression in a formal query language, which returns the intended answer $a \in A$ when executed over $K$. Here, $A$ is the set of all answers a KGQA system can be expected to retrieve, consisting of (i) a subset of entities ($e_i$) or literals ($l_i$) from $K$, (ii) the result of an arbitrary aggregation function ($f : \{e_i\} \cup \{l_i\} \mapsto \mathbb{N}$), or (iii) a boolean ($T/F$) variable depending on whether the subgraph implied by $Q$ is a subset of $K$.

In contrast, answering *simple questions* is a subset of the above KGQA task where (i) the answer set can only contain $A$, a subset of entities ($e_i$) or literals ($l_i$), and (ii) members of the answer set must be directly connected to the topic entity with a predicate $p_i \in \mathcal{P}$. This work aims to solve the aforementioned KGQA task, which implicitly includes answering simple questions as well.




(a) *Question, and corresponding Query Graph* (b) *Core-chain corresponding to (a)*

Fig. 1: A question and its corresponding query graph (a), and core chain (b).

### 3.2 Query Graphs

We use query graphs as the intermediary query language to express candidates of formal KG queries given an NLQ. They represent a path in $K$ as a directed acyclic labeled graph. We borrow the augmentations made to the query graph grammar in [23], which makes the conversion from query graph expressions to executable queries trivial, and slightly modify it to suit our use case. In the subsequent paragraphs we detail the modified query graph grammar we employ.

*Elements of a query graph:* A query graph consists of a combination of nodes $n \in \{$*grounded entity*, *existential variable*, *lambda variable*, *auxiliary function*$\}$, connected with labeled, directed edges representing predicates from the set of predicates $\mathcal{P}$ of $K$. We define each of the aforementioned elements in detail below by the help of a running example, answering the question "*Name some movies starring Beirut born male actors?*":

- **Grounded Entities**: Grounded entities correspond to entities of $K$ mentioned in the NLQ. Each query graph has at least one grounded entity. In the case of our example, ex:Beirut is the grounded entity represented by the rounded rectangle in Fig. 1.a.
- **Existential Variables**: Existential variables are ungrounded nodes, i.e. they do not correspond to an explicit entity mentioned in the NLQ but are instead placeholders for intermediate entities in the KG path. They help disambiguate the structure of query graphs by the means of auxiliary functions (described below). In our example, we have one existential variable which is represented by a circle and stands for entities like ex:Keanu_Reeves and ex:Nadine_Labaki.
- **Lambda Variables**: Similar to *existential variables*, they are ungrounded nodes acting as a placeholder for a set of entities which are the potential answer to the query. The are represented by shaded circles in our example, and can have entities like ex:John_Wick (a movie) and ex:Rain (a TV series) mapped to it.
- **Auxiliary Functions**: Auxiliary functions are applied over the set of entities mapped to any ungrounded node (i.e. *grounded* and *existential variables*) in the query graph. In our grammar, they can be of two types, namely, the cardinality function or a class constraint $f_{class} : \{e \in E | (e,\ \text{rdf:type},\ class) \in K\}$ where $(class, \text{rdf:type}, \text{owl:Class}) \in K$. In our example, we can apply the class constraint function over the entities mapped to the existential variable to only include male actors, and to the entities mapped to the lambda variable, represented by rectangle, to only include movies. The cardinality constraint can be used for NLQs like "*How many*

*movies have casted Beirut born male actors?*" by imposing a count constraint over the lambda variable.

Finally, a new flag is defined which determines whether the query graph is used to fetch the value of the lambda variable, or to verify whether the graph is a valid subset of the target KG. The latter is used in the case of Boolean queries like "*Did Keanu Reeves act in John Wick?*" We represent this decision with a flag instead of another node or constraint in the graph as it doesn't affect the execution of the query graph, but in the case of a Boolean query only inquires, post execution, whether the query had a valid solution.

*Query Graph Representation:* We represent the query graphs in a linear form so as to easily encode them in our ranking models. We linearize the directed graph by starting from one of the grounded entities and using $+, -$ signs to denote the outgoing and incoming edges, respectively. We represent auxiliary functions with another flag along with the linearized chain. Externalizing the auxiliary functions enables us to remove the ungrounded nodes from the core chain, which is now composed of the grounded entity and relations prefixed by $+, -$ signs. Finally, we replace the URIs of the grounded entities and predicates with their corresponding surface forms. Hereafter, we refer to this linearized representation as the *core chain* of a query graph. This representation also ensures that the query graph maintains textual similarity to the source NLQ, enabling us to use a wide variety of text similarity based approaches for ranking them. Fig 1.b illustrates the core chain corresponding to the query graph in our running example. In our preliminary analysis we found that removing mentions of grounded entities in the core chain increased the performance of our approach. We thus exclude the grounded entities from the final core chain representation. Since the grounded entities are the same for a given NLQ, no information is lost from the core chain candidate set upon doing this.

## 4 Approach Overview

We treat KGQA as the task of generating and ranking query graph candidates w.r.t. a given NLQ. For instance, given the question "What is the population of the capital of Germany?", we would like a ranking model to assign a higher score to "`+ capital + population`" than "`+ capital + mayor`", where "+" indicates that the relation must be followed in the forward direction. More formally, given a question $Q$ and a set of candidate core chains $C^1 \dots C^N$, we select the most plausible core chain as follows:

$$C^* = \underset{C^i}{\operatorname{argmax}}\, \mathsf{sim}(Q, C^i) \ , \tag{1}$$

where $\mathsf{sim}(\cdot, \cdot)$ is a function assigning a score to a pair of a NLQ and a core chain. We implement $\mathsf{sim}(\cdot, \cdot)$ as the dot product of two vectors produced by the encoder $\mathsf{enc}^{\mathsf{q}}(Q)$ and the core chain encoder $\mathsf{enc}^{\mathsf{c}}(C^n)$ respectively, i.e.,

$$\mathsf{sim}(Q, C^n) = \mathsf{enc}^{\mathsf{q}}(Q) \cdot \mathsf{enc}^{\mathsf{c}}(C^n) \ . \tag{2}$$

We train our ranking model with a pairwise loss function that maximizes the difference between the score of correct (positive) and incorrect (negative) pairs of NLQs and core chains, that is

$$L = \max(0, \gamma - \mathsf{sim}(Q, C^+) + \mathsf{sim}(Q, C^-)) \ , \tag{3}$$

where $\mathsf{sim}(Q, C^+)$ and $\mathsf{sim}(Q, C^-)$ are the scores for the correct and incorrect question-core chain pair, respectively.

We assume the entities mentioned in the NLQ to be given (but do not require exact entity spans i.e which tokens in the question correspond to which entity). In the next section (Sec. 4.1), we outline a mechanism for generating core chain candidates. Following that, we describe a novel core chain ranking model in Sec. 4.2. Furthermore, for a fully functioning QA system, additional auxiliary functions needs to be predicted. We define them, and outline our method of predicting them in Sec. 4.3.

### 4.1 Core Chain Candidate Generation

Core chains, as described in the previous section, are linearized subsets of query graphs which represent a path consisting of entities and predicates without the additional constraints. Working under the assumption that the information required to answer the question is present in the target KG, and that we know the entities mentioned in the question, we collect all the plausible paths of up to two hops from an arbitrary grounded entity node[4] to generate the core chain candidate set. Here, we use the term *hop* to collectively refer to a KG relation along with the corresponding $+/-$ sign indicating whether the relation is incoming or outgoing w.r.t. the entity.

We retrieve candidate core chains by collecting all predicates (one-hop chains) and paths of two predicates (two-hop chains) that can be followed from an arbitrary grounded node. In this process, predicates are followed in both outgoing and incoming direction (and marked with a $+$ and $-$ in the chain, respectively). We further restrict our candidate set of core chains as follows: if two entities have been identified in the question, we discard the core chains which do not contain both the entities as grounded nodes. When applied, this step substantially decreases the candidate set while retaining all the relevant candidates. Finally, we drop the mention of entities from the core chain since every core chain thus generated will contain the same entities in the same position, and doing so leads to no information loss. Doing so enables our ranking models to retain the focus on comparing the predicates of the core chain to the question.

Although we limit the core chains to a length of two hops for the purposes of this study, this approach can easily be generalized to longer core chains. However, it may result in an additional challenge of handling a larger number of candidate core chains.

### 4.2 Slot Matching Model

To exploit the specific structure of the task, we propose an encoding scheme which partitions core chains into the aforementioned *hops*, and creates multiple, hop-specific representations of the NLQ, which we call *slots*. We then compare the hop (segments of

[4] Entity that has been linked in the question.

a core-chain) representations with their corresponding slot (an encoded representation of the NLQ) to get the final score.

First, the question $Q = \{q_1 \dots q_T\}$ is encoded using a bidirectional LSTM ($\mathsf{LSTM}^q$) resulting in the question encoding

$$[\hat{\mathbf{q}}_1 \dots \hat{\mathbf{q}}_T] = \mathsf{LSTM}^q(Q) \ . \tag{4}$$

Now, consider a core chain consisting of $M$ hops. For each hop $j = 1, \dots, M$, we define a trainable slot attention vector $\mathbf{k_j}$ which is used to compute attention weights $\alpha_{t,j}$, individually for every hop $j$, over all the words $q_t, t = 1, \dots, T$ of $Q$. Then, a set of fixed-length question representations $\mathbf{q}_j$ are computed using the corresponding attention weights $\alpha_{t,j}$, that is

$$\alpha_{t,j} = softmax(\{< \hat{\mathbf{q}}_l, \mathbf{k}_j >\}_{l=1\dots T})_t \quad , \tag{5}$$

$$\mathbf{q}_j = \sum_{t=1}^{T} \alpha_{t,j} \cdot \hat{\mathbf{q}}_t \ . \tag{6}$$

We represent the core chains by separately encoding each hop by another LSTM ($\mathsf{LSTM}^c$)

$$\mathbf{c}_j = \mathsf{LSTM}^c(C_j) \ , \tag{7}$$

where $C_j = [c_{j,1} \dots c_{j,T'_j}]$ is the sequence of words in the surface from of the predicate along with the $+/-$ signs, corresponding to the $j^{th}$ hop of the core chain. Finally, $\mathbf{q}_1, \dots, \mathbf{q}_M$ and $\mathbf{c}_1, \dots, \mathbf{c}_M$ are concatenated to yield our final representation of the NLQ and the query graph ($\mathsf{enc}^{\mathsf{q}}(Q)$ and $\mathsf{enc}^{\mathsf{c}}(C)$), respectively, which is used in score function given in Eqn. 2, i.e.

$$[\mathbf{q}_1, \dots \mathbf{q}_M] = \mathsf{enc}^{\mathsf{q}}(Q) \tag{8}$$

$$[\mathbf{c}_1, \dots \mathbf{c}_M] = \mathsf{enc}^{\mathsf{c}}(C) \ . \tag{9}$$

Figure 2 summarizes the proposed approach.

Note that the model proposed here is not the same as *cross attention* between the input sequences (as described by [16] which we also experiment with) as, in our case the attention weights aren't affected by the predicates in the core chain, as the encoder attempts to focus on *where* a predicate is mentioned in $Q$, and not *which* predicate is mentioned. In Sec 5.1, we discuss advantages of *slot based attention* over *cross attention* in further detail.

**Using Transformers in the Slot Matching configuration:** [6] demonstrate that the use of pre-trained bidirectional transformers (BERT) can provide improvements for numerous downstream NLP tasks. Motivated by their findings, we investigate whether they can positively impact the performance on our KGQA task as well.

In this subsection, we describe how we use BERT to encode the NLQ and the core chains in the slot matching model. In the simplest approach, we would simply replace the $\mathsf{LSTM}$ in Eqn. (4) and (7) with pre-trained transformers and keep the rest of the model unchanged.

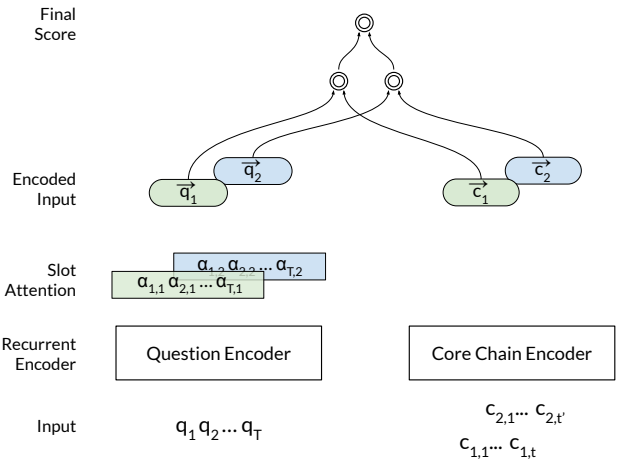



Fig. 2: The slot matching model uses parameterized attention vectors to create $j$ representations of the question, and compares each of them correspondingly with the $j$ hops in a core chain. Here $t$ and $t^{'}$ represent the number of words in each hop of the core chain, and $c_{t,1}$ is the $t^{th}$ word in the first hop.

However, [6, 18] prescribe converting structured inputs into an single ordered sequence. We thus concatenate our inputs: (i) a question $Q = [q_1 \dots q_T]$ of length $T$, and (ii) the $M$ hops of a core chain $C = \big[\big[c_{1,1} \dots c_{1,T_1'}\big] \cdots \big[c_{M,1} \dots c_{M,T_M'}\big]\big]$ into a sequence of length $l = T + \sum_{j=1}^{M} T_j'$ (excluding sequence delimiters), and pass it through the transformer. Concretely, we use [6]'s input encoding scheme: we (1) prepend the sequence with a `[CLS]` token, (2) append the `[SEP]` separator token at the end of the question and (3) separate the different predicate surface forms in the appended candidate core chain with the same `[SEP]` token. The input to the transformer corresponding to our previous example then looks like this: "`[CLS]` *Name some movies starring Beirut born male actors* `[SEP]` + *capital* `[SEP]` + *population* `[SEP]`". [6] use the output of the transformer at first position (corresponding to the `[CLS]` token) for classification. Instead, for our slot matching transformer, we replace $[\hat{\mathbf{q}}_1 \dots \hat{\mathbf{q}}_T]$ in eq. (4) with the question portion of the transformer's outputs. Applying eq. (5) and (6) as before yields a set of slot-specific question encodings $\mathbf{q}_1, \dots, \mathbf{q}_M$. Slot-specific hop encodings $\mathbf{c}_1, \dots, \mathbf{c}_M$ are obtained from the same sequence of output vectors of the transformer by taking the representation at the `[SEP]` delimiter preceding the $j^{th}$ hop. Given these encodings, the score for a question-chain pair is computed as before. The model is depicted in Fig 3.

### 4.3 Predicting Auxiliary Functions

In this section, we describe our approach towards learning to predict the auxiliary functions used for constructing a complete query graph. We begin by predicting the intent of the question. In both the datasets considered in our experiments, a question can ask for the cardinality of the lambda variable, ask whether a certain fact exists in the KG, or

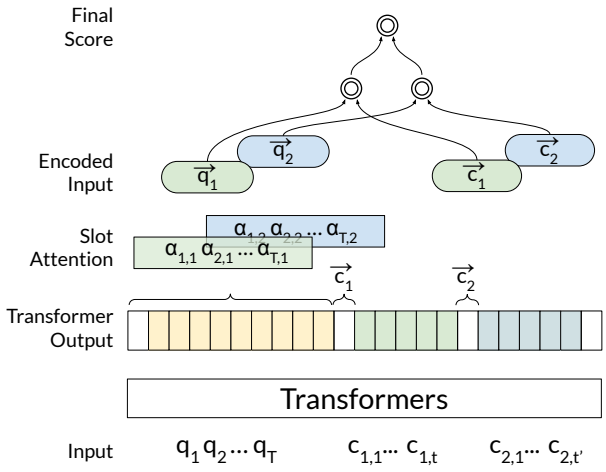



Fig. 3: Illustration of the transformer model in the slot matching configuration.

simply ask for the set of values in the lambda variable. Further, this division, hereafter referred to as *count*, *ask* and *set* based questions, is mutually exclusive. We thus use a simple BiLSTM based classifier to predict the intent belonging to one of the three classes.

Next, we focus on detecting class based constraints on the ungrounded nodes of the core chain, as described in Sec.3.2 as $f_{class}$. We use two different, separately trained models to predict (i) whether such a constraint exists in the NLQ, and if so, on which variable, and (ii) which $class$ is used as a constraint. The former is accomplished with a simple binary BiLSTM classifier (i.e. constraint exist or not), similar to the aforementioned intent classifier. For the latter, we use a BiLSTM based pairwise ranking model trained in a similar setting as described in Eqn. 3.

We now have all the information required to construct the query graph, and the corresponding executable query. For brevity's sake, we omit the algorithm to convert query graphs to SPARQL here, but for limited use cases such as ours, simple template matching (based on the $+/-$ signs of the selected core chain, the class constraint, and the result of the intent classifier) shall suffice.

## 5 Experiments

In a first set of experiments we compare the KGQA performance of the proposed slot matching model with some baseline models as described in the following section. After that, we describe experiments investigating transfer learning across KGQA datasets and from pre-trained transformers.

### 5.1 Approach Evaluation

Our first experiment focuses on investigating the performance of the **Slot Matching (LSTM)** model. As a baseline we use a simple neural ranking model where we replace

$\text{enc}^q$ and $\text{enc}^c$ with a single layered bidirectional LSTM (**BiLSTM**). We also compare our model to those proposed by [16] (decomposable attention model, or **DAM**), [24] (hierarchical residual model, or **HRM**), and [11] which uses a multi-channel convolutional neural network (**CNN**).

**Datasets** Our models are trained and evaluated over the following two KGQA datasets:
**LC-QuAD** [20] is a gold standard question answering dataset over the DBpedia 04-2016 release, having 5000 NLQ and SPARQL pairs. The coverage of our grammar covers all kinds of questions in this dataset.
**QALD-7** [21] is a long running challenge for KGQA over DBpedia. While currently its $9^{th}$ version is available, we use QALD-7 (Multilingual) for our purposes, as it is based on the same DBpedia release as that of LC-QuAD. QALD-7 is a gold-standard dataset having 220 and 43 training and test questions respectively along with their corresponding SPARQL queries. Some of the questions in the dataset are outside the scope of our system. We nonetheless consider all the questions in our evaluation.

**Evaluation Metrics** We measure the performance of the proposed methods in terms of their ability to find the correct core chain, as well as the execution results of the whole system. For core chain ranking, we report Core Chain Accuracy (CCA) and Mean Reciprocal Rank (MRR). Based on the execution results of the whole system (including auxiliary functions), we also report Precision (P), Recall (R), and F1.

**Training** Our models are trained with negative sampling, where we sample 1000 negative core chains per question, along with the correct one, for every iteration. We train our models for a maximum of 300 epochs, using a 70-10-20 split as train, validation and test data over LC-QuAD [5] QALD-7 has a predefined train-test split, and we use one eighth of the train data for validation. We embed the tokens using Glove embeddings [17], and keep the relevant subset of the embeddings trainable in the model. We use Adam optimizer with an initial learning rate of 0.001 and set the margin ($\gamma$) in the pairwise loss function as 1.

We share parameters between $\text{enc}^q$ and $\text{enc}^c$ in the BiLSTM, CNN and DAM models, since in these models, input sequences are processed in the same manner, while the same does not hold for the slot matching, or HRM. We illustrate the impact of this choice on model performance towards the end of this section.

**Results** In our experiments, the slot matching model performs the best among the ones compared, suggesting that different attention scores successfully create suitably weighted representations of the question, corresponding to each hop. This is further reinforced upon visualizing attention scores, as presented in Fig 4, where we notice that

[5] That is, we use the first 70% of dataset, as made available on `https://figshare.com/projects/LC-QuAD/21812` by [20], to train our models. Next 10% is used to decide the best hyperparamters. The metrics we report in the rest of this section are based on the models performance on the last 20% of it.

| | LC-QuAD | | | | | QALD-7 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CCA | MRR | P | R | F1 | CCA | MRR | P | R | F1 |
| BiLSTM [9] | 0.61 | 0.70 | 0.63 | 0.75 | 0.68 | 0.28 | 0.41 | 0.20 | 0.36 | 0.26 |
| CNN [11] | 0.44 | 0.55 | 0.49 | 0.61 | 0.54 | 0.31 | **0.45** | 0.20 | 0.33 | 0.25 |
| DAM [16] | 0.57 | 0.66 | 0.59 | 0.72 | 0.65 | 0.28 | 0.40 | 0.20 | 0.36 | 0.26 |
| HRM [24] | 0.62 | 0.71 | 0.64 | 0.77 | 0.70 | 0.28 | 0.40 | 0.15 | 0.31 | 0.20 |
| **Slot-Matching (LSTM)** | **0.63** | **0.72** | **0.65** | **0.78** | **0.71** | **0.31** | 0.44 | **0.28** | **0.44** | **0.34** |

Table 1: Performance on LC-Quad and QALD-7. The reported metrics are core chain accuracy (CCA), mean reciprocal rank (MRR) of the core chain rankings, as well as precision (P), recall (R), and the F1 of the execution results of the whole system.

the different attention *slots* focus on different predicate spans in the question. While the decomposable attention model (DAM) proposed by [16] also uses attention, its performance generally lags behind the slot matching model. DAM's cross-attention between question and core chain leads to a summarized question representation that is dependent on the candidate core chain and vice versa. On the other hand, the slot matching approach merely attempts to learn to extract important, relation-specific parts of the NLQ, prior to seeing a specific core chain, which judging by our experiments seems to be a better model bias and might help generalizing. The hierarchical residual model (HRM) [24] is second best in our comparison, suggesting that pooling relation and word level encodings is a promising strategy to form core chain representations. The competitive performance of the BiLSTM model is in coherence with recent findings by [15], that a simple recurrent model can perform almost as well as the best performing alternative.

All models generally exhibit poor performance over QALD-7, which is understandable given the fact that QALD-7 has only 220 examples in the training set, which is 20 times smaller than LC-QuAD. We will show in the next section that transfer learning across datasets is a viable strategy in this case to improve model performance.

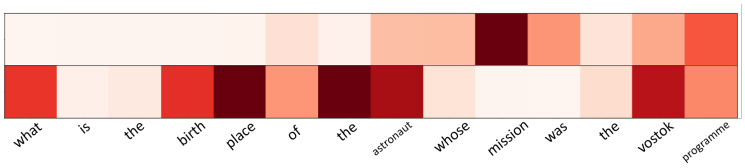



Fig. 4: Visualized attention weights (darker color corresponds to larger attention weights) of the slot matching question encoder for the question "*What is the birth place of the astronaut whose mission was the vostok programme?*" The two rows represent different slot attention scores. One can see that first puts a higher weight on *mission* while the second (beside others) on *birth place*.

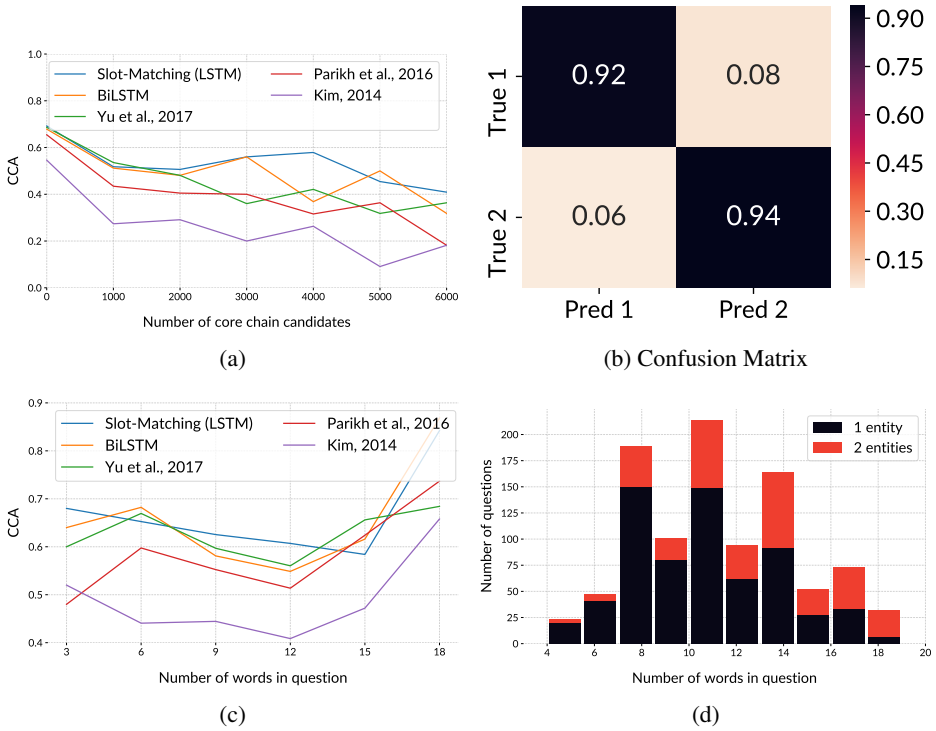



Fig. 5: Here, (a) shows the decrease in accuracy with increasing number of candidates for all the rankings models in Section 5.1; (b) is a confusion matrix representing the number of hops in true and predicted core chains for the test data of LC-QuAD for the slot matching model. (c) the relation of model accuracy w.r.t. question length. And (d) a histogram depicting the distribution of questions in LC-QuAD's test split w.r.t. their question lengths. Here, the proportion of questions with two entity mentions are depicted with red color.

**Error Analysis:** We now illustrate the effect of different characteristics of the test data on the model performance.

**Effect of number of core chain candidates**: In our ranking based approach, the difficulty of selecting the correct core chain depends on the number of core chain candidates, which can be disproportionately large for questions about well-connected entities in DBpedia (e.g. `dbr:United_States`). In order to investigate its effect, we plot the core chain accuracy (CCA) vs. the number of core chain candidates, for all the models we experiment with, in Fig. 5a. Upon inspection, we find the core chain accuracy to be inversely correlated to the number of core chain candidates. Specifically, we find that the performance of the BiLSTM, HRM and the slot pointer model (the three best performing ones in Exp. 5.1) remain almost the same for as many as 2000 core chain candidates per question. Thereafter, the BiLSTM and HRM models' accuracy declines

faster than that of the proposed slot matching model, giving a competitive edge to the latter.

**Effect of length of questions**: We noticed that it is relatively easier for the models to answer longer questions. To better understand this phenomenon, we plot the core chain accuracy w.r.t. the length of questions for all the models in Fig. 5c, and the frequency of questions w.r.t. their lengths in Fig. 5d.

We find that longer questions are more likely to contain two entity mentions than just one. This hints to the fact that the number of candidate core chains reduces accordingly, as every valid core chain candidate must include all entity mentions of the question, which simplifies the ranking process as detailed in Sec. 4.1.

**Effect of length of core chains**: Inherent biases in the data might make our models more inclined to assign higher ranks to paths of a certain length, at the expense of selecting the correct path. We thus compose a confusion matrix representing the number of hops in the ground-truth and predicted core chains, which we hypothesize can help detect these biases. We find that none of our models suffer from this issue. As an example, we visualize the confusion matrix for the slot matching model's predictions over LC-QuAD's test split in Fig. 5b.

**Further Analysis:** In order to better assess the impact of different parts of the system we perform a series of analysis over the simplest baseline (BiLSTM), and the best performing model (Slot Matching (LSTM)). For brevity's sake we only report the core chain accuracy in these experiments. Unless specified otherwise, the hyperparameters will be the same as mentioned in the experiment above.

**Ablation Study**: In order to better understand the effect of slot-specific attention over the question in the slot matching model, we experiment with a simpler model where we use the same attention scores for each slot. Effectively, this transforms our $\mathsf{enc}^{\mathsf{q}}$ to a simpler, single-slot attention based encoder.

In our experiments, we find that the model yields similar results to that of BiLSTM model, i.e. 60.3%, which is considerably worse (- 2.8%) than the regular slot matching model with two slots. Our experiments illustrate several mechanism of using attention for the task, including no attention (BiLSTM, 61.4%), with attention (single slot, 60.3%), with multiple slots of attention (slot matching model, 63.1%), and with cross attention (DAM, 56.8%).

**Parameter Sharing between Encoders**: In the primary experiment, the BiLSTM model shares parameters between

$\mathsf{enc}^{\mathsf{q}}$ and $\mathsf{enc}^{\mathsf{c}}$, while the slot matching model doesn't. To show the effect of parameter sharing between encoders, we retrain both models in both settings (with and without parameter sharing).

Sharing encoders leads to a *decrease* of 2.9% (60.4% from 63.1%) in CCA of the slot matching model. Conversely, sharing encoders *increases* the performance of the BiLSTM model by 3.1% (61.4% from 58.3%). In the BiLSTM's case, learning a mapping that captures the equivalence of questions and core chains is not hindered by sharing parameters because the model structure is the same on both sides (simple encoders). Sharing parameters in this case could help because of the decrease in the total number of parameters. In the case of the slot matching model, however, sharing the parame-

ters of the encoders would require the encoder to be usable for both the attention-based summary of the question encoder as well as the simple encoder for each hop (where the latter processes much shorter sequences) which leads to a performance bottle neck.

### 5.2 Transfer Learning across KGQA datasets

As mentioned above, all models generally show poor performance when trained solely on QALD-7 due to a more varied and significantly smaller dataset. Therefore, we hypothesize that pre-training the models on the much larger LC-QuAD dataset might lead to a significant increase in performance. To that end, we perform the following fine-tuning experiment: we pre-train our ranking models over LC-QuAD, and then fine-tune and evaluate them over QALD-7. We set the initial learning rate to 0.0001 (which is an order of magnitude less than in the experiments in Section 5.1), and experiment with custom learning rate schedules, namely *slanted triangular learning rate* (sltr) proposed in [10], and *cosine annealing* (cos) proposed in [13]. We keep the hyperparameters of *sltr* unchanged, and set the number of cycles for *cos* to 3 based on the performance on the validation set.

| Learning Rate | BiLSTM | Slot Matching (LSTM) |
|---|---|---|
| constant | 0.37 | 0.37 |
| **sltr** | **0.39** | **0.42** |
| cos | 0.25 | 0.28 |

Table 2: CCA for the fine-tuning experiment where we pre-train on LC-QuAD and fine-tune over QALD-7. The initial learning rate is $10^{-3}$ for all configurations.

We conduct the experiment on the BiLSTM and the Slot Matching (LSTM) ranking model, and only report the *core chain accuracies* (CCA) as the rest of the system remains unchanged for the purposes of this experiment.

We find that *fine-tuning* a ranking model trained over LC-QuAD leads to a substantial ($\sim 11\%$) increase in performance on QALD-7 compared to non-pre-trained models. Interestingly, we find that the results of the fine-tuning experiment are sensitive to the learning rate schedule used. While constant learning rate provides a relatively comparable performance w.r.t. *sltr*, using the cosine annealing schedule adversely affects model performance.

We report the results of this experiment in Table 2. In summary we conclude that transferring models across KGQA datasets via simple fine-tuning is a viable strategy to compensate for the lack of training samples in the target dataset.

### 5.3 Transfer Learning with pre-trained Transformers

For our transformer based slot matching model, we use a transformer, initialized with the weights of BERT-Small[6], instead of an LSTM, as discussed in Sec 4.2. The transformer has 12 layers of hidden size 768 and 12 attention heads per layer. Following [6], we set dropout to 0.1. We train using Adam with initial learning rate 0.00001. Table 3 shows the performance of the pre-trained transformer (**BERT**), used as in [6] as well as the pre-trained transformer in the slot matching configuration (**Slot Matching (BERT)**). For BERT, we follow the sequence pair classification approach described by [6].

| | QALD-7 | LC-QuAD |
|---|---|---|
| BERT | **0.23** | 0.67 |
| Slot Matching (BERT) | 0.18 | **0.68** |

Table 3: CCA for slot matching model, as proposed in Sec 4.2 initialized with the weights of BERT-Small, compared with regular transformers initialized with the same weights.

Through this experiment we find that using pre-trained weights immensely improves model performance, as both transformer based models outperform the ones in Section 5.1. Additionally, we find that the augmentations we propose in Sec 4.2 are beneficial for the task, improving CCA on LC-QuAD by 1.4% relative to regular pre-trained transformers. However, both models exhibit poor performance over QALD-7, suggesting that these models need substantial amounts of data to be properly fine-tuned for the task. We thus conclude that using pre-trained transformers in the slot matching setting is advantageous for the task, if ample training data is available at hand.

## 6 Conclusion and Future Work

In this work, we studied the performance of various neural ranking models on the KGQA task. First, we propose a novel task-specific ranking model which outperforms several existing baselines in our experiments over two datasets. An error analysis shows that the model performs especially well on smaller candidate sets and for longer questions which highlights its high potential for answering complicated questions. Second, we present an extensive study of the use of transfer learning for KGQA. We show that pre-training models over a larger task-specific dataset and fine-tuning them on a smaller target set leads to an increase in model performance. We thereby demonstrate the high potential of these techniques for offsetting the lack of training data in the domain. Finally, we propose mechanisms to effectively employ large-scale pre-trained state of the art language models (BERT [6]) for the KGQA task, leading to an impressive performance gain on the larger dataset.

[6] as provided by the authors at https://github.com/google-research/bert

We aim to extend this work by incorporating a three phased domain adaption strategy as proposed in [10] for the KGQA task. We will study the effect of pretraining our ranking models with synthetically generated datasets, aiming for consistent coverage of relations involved in the ranking process. Further, we intend to explore differentiable formal query execution mechanisms [4] enabling answer supervision based training of our model.

**Acknowledgements** This work has been supported by the Fraunhofer-Cluster of Excellence "Cognitive Internet Technologies" (CCIT).

## References

1. Bast, H., Haussmann, E.: More accurate question answering on freebase. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. pp. 1431–1440. ACM (2015)
2. Berant, J., Liang, P.: Semantic parsing via paraphrasing. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1415–1425. Association for Computational Linguistics (2014). https://doi.org/10.3115/v1/P14-1133, http://www.aclweb.org/anthology/P14-1133
3. Bowman, S.R., Angeli, G., Potts, C., Manning, C.D.: A large annotated corpus for learning natural language inference. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics (2015)
4. Cohen, W.W.: Tensorlog: A differentiable deductive database. arXiv preprint arXiv:1605.06523 (2016)
5. Cui, W., Xiao, Y., Wang, H., Song, Y., Hwang, S.w., Wang, W.: Kbqa: Learning question answering over qa corpora and knowledge bases. Proc. VLDB Endow. **10**(5), 565–576 (Jan 2017). https://doi.org/10.14778/3055540.3055549, https://doi.org/10.14778/3055540.3055549
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
7. Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., Lehmann, J.: Asknow: A framework for natural language query formalization in sparql. In: International Semantic Web Conference. pp. 300–316. Springer (2016)
8. Fader, A., Zettlemoyer, L., Etzioni, O.: Open question answering over curated and extracted knowledge bases. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1156–1165. ACM (2014)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
10. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). vol. 1, pp. 328–339 (2018)
11. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
12. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al.: Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web **6**(2), 167–195 (2015)
13. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)

14. Lukovnikov, D., Fischer, A., Lehmann, J.: Pretrained transformers for simple question answering over knowledge graphs. In: International Semantic Web Conference. Springer (2019)
15. Mohammed, S., Shi, P., Lin, J.: Strong baselines for simple question answering over knowledge graphs with and without neural networks. arXiv preprint arXiv:1712.01969 (2017)
16. Parikh, A., Täckström, O., Das, D., Uszkoreit, J.: A decomposable attention model for natural language inference. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 2249–2255. Association for Computational Linguistics (2016). https://doi.org/10.18653/v1/D16-1244, http://www.aclweb.org/anthology/D16-1244
17. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (2014), http://www.aclweb.org/anthology/D14-1162
18. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training. URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf (2018)
19. Reddy, S., Täckström, O., Petrov, S., Steedman, M., Lapata, M.: Universal semantic parsing. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 89–101. Association for Computational Linguistics (2017), http://aclweb.org/anthology/D17-1009
20. Trivedi, P., Maheshwari, G., Dubey, M., Lehmann, J.: Lc-quad: A corpus for complex question answering over knowledge graphs. In: International Semantic Web Conference. pp. 210–218. Springer (2017)
21. Usbeck, R., Ngomo, A.C.N., Haarmann, B., Krithara, A., Röder, M., Napolitano, G.: 7th open challenge on question answering over linked data (qald-7). In: Semantic Web Evaluation Challenge. pp. 59–69. Springer (2017)
22. Xu, K., Zhang, S., Feng, Y., Zhao, D.: Answering natural language questions via phrasal semantic parsing. In: Natural Language Processing and Chinese Computing, pp. 333–344. Springer (2014)
23. Yih, W.t., Chang, M.W., He, X., Gao, J.: 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). vol. 1, pp. 1321–1331 (2015)
24. Yu, M., Yin, W., Hasan, K.S., dos Santos, C., Xiang, B., Zhou, B.: Improved neural relation detection for knowledge base question answering. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 571–581. Association for Computational Linguistics (2017). https://doi.org/10.18653/v1/P17-1053, http://www.aclweb.org/anthology/P17-1053