Crake: Causal-Enhanced Table-Filler for Question Answering over Large Scale Knowledge Base

Minhao Zhang[1♣] **Ruoyu Zhang**[1♣] **Yanzeng Li**[1♦] **Lei Zou**[1,2♣]
[1] Wangxuan Institute of Computer Technology (WICT), Peking University, China;
[2]Beijing Academy of Artificial Intelligence, Beijing, China;
♣{zhangminhao,ry_zhang,zoulei}@pku.edu.cn
♦ liyanzeng@stu.pku.edu.cn

Abstract

Semantic parsing solves knowledge base (KB) question answering (KBQA) by composing a KB query, which generally involves node extraction (NE) and graph composition (GC) to detect and connect related nodes in a query. Despite the strong causal effects between NE and GC, previous works fail to directly model such causalities in their pipeline, hindering the learning of subtask correlations. Also, the sequence-generation process for GC in previous works induces ambiguity and exposure bias, which further harms accuracy. In this work, we formalize semantic parsing into two stages. In the first stage (graph structure generation), we propose a causal-enhanced table-filler to overcome the issues in sequence-modelling and to learn the internal causalities. In the second stage (relation extraction), an efficient beam-search algorithm is presented to scale complex queries on large-scale KBs. Experiments on LC-QuAD 1.0 indicate that our method surpasses previous state-of-the-arts by a large margin (17%) while remaining time and space efficiency. The code and models are available at https://github.com/AOZMH/Crake.

1 Introduction

To incorporate knowledge in real-world question-answering systems, knowledge base question answering (KBQA) utilizes a background knowledge base (KB) as the source of answers to factoid natural language questions. Leveraging the versatility of KB query languages like SPARQL (Prud'hommeaux, 2008), many previous works (Unger et al., 2012; Yahya et al., 2012) adopted a semantic parsing paradigm for KBQA, in which questions are converted to equivalent SPARQL queries and answers are given by executing the queries in KB. Regarding the intrinsic graph structure of SPARQLs, some works further reduced such procedure as generating the query graph of SPARQLs w.r.t. questions. However, these methods either require auxiliary tools (e.g. AMR in Kapanipathi et al., 2021, constituency tree in Hu et al., 2021, dependency tree in Hu et al., 2017) causing potential cascading errors, or rely on predefined templates (Cui et al., 2017; Athreya et al., 2021) limiting their expressiveness and generalization abilities.

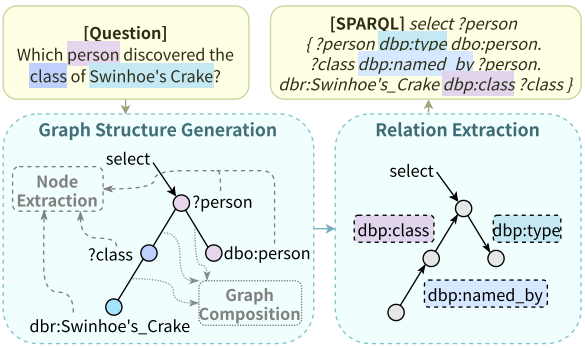

Figure 1: Generating a query graph (bottom) by two stages to represent the SPARQL (right-top). At graph structure generation stage, node-extraction generates all graph nodes while graph-composition adds unlabeled edges between proper nodes. Then, the relation extraction stage decides the specific predicate of each edge.

To address these, efforts were made on devising independent pipelines for query graph construction (Lin et al., 2021). As in Figure 1, these pipelines usually involve a node extraction (NE) module to detect the mentions of all nodes in query graph and link entity mentions, a graph composition (GC) module to connect related nodes given by NE, and a relation extraction (RE) module deciding the KB predicate corresponding to each edge added in GC. In this framework, two drawbacks exist in previous works: 1) we observe strong causal effects between NE and GC, e.g. edges connected by GC are valid only between the node mentions extracted in NE, making GC decisions highly dependent on NE. To this regard, previous works (Zhang et al., 2021; Ravishankar et al., 2021) that perform NE and GC separately without causal-modelling may fall short in deeply comprehending the correlated tasks

and accurately generating query graphs. 2) GC is commonly modelled as a sequence-generation in prior methods, either through generative decoder (Shen et al., 2019; Chen et al., 2021) or via stage-transition (Yih et al., 2015; Hu et al., 2018). However, sequence-modelling generally undergoes sequence ambiguity and exposure bias (Zhang et al., 2019) that harms model accuracy.

In this work, we formalize the generation of query graph in a two-staged manner as in Figure 1. At the first stage, we tackle the aforesaid weaknesses by a novel causal-enhanced table-filling model to jointly complete NE and GC, resulting in a query graph structure representing the connectivity of all nodes. More specifically, inspired by Chen et al. (2020a), we utilize a label transfer mechanism to facilitate the acquisition of causality between NE and GC (which solves drawback 1 above). Further, we apply a table-filler to decode all edges simultaneously, which naturally circumvents the ambiguity and bias of iterative decoding (and solves drawback 2). For the second stage, we propose a beam-search-based relation extraction algorithm to determine the predicate that binds to each graph edge. Differ from prior works, we perform candidate predicate retrieval and ranking alternately for each edge, limiting the candidate scale linearly w.r.t. KB degree and making the algorithm scalable for large-scale KBs like DBpedia.

In short, the major contributions of this paper are: 1) to our knowledge, we are the first to model GC as a table-filling process, which prevents the ambiguity and bias in prior works; 2) we model the intrinsic causal effects in KBQA to grasp subtask correlations and improve pipeline integrity; 3) our method outperforms previous state-of-the-arts on LC-QuAD 1.0, a prominent KBQA benchmark, by a large margin (~17%), further experiments verifies the effectiveness of our approach.

2 Preliminaries

2.1 Problem Setting

We solve KBQA in a semantic parsing way, given a question (left-top in Figure 1), we generate a SPARQL query (right-top in Figure 1) to represent its semantics and answer the question by executing the query in KB. By definition, SPARQL describes a query graph with each triple in its body referring to a graph edge; by matching the graph pattern in KB, certain KB entries binding to the query graph can be processed as query results (e.g. in Table 1 for SELECT queries, all entries binding to the "select" node are results; for JUDGE queries, the existence of matched entries determines the boolean result). Hence, our task is further specified as constructing the query graph (bottom of Figure 1) of a question to represent its corresponding SPARQL.

Type	Example SPARQL
JUDGE	ask {dbr:New_York a dbo:City}
COUNT	select count(?x) {?x a dbo:City}
SELECT	select ?x {?x a dbo:City}

Table 1: Supported query types.

2.2 Methodology Overview

Illustrated by Figure 1, we construct the query graph in two stages. In the graph structure generation stage (bottom-left in Figure 1), we extract all graph nodes by finding the mention of each node in question and its tag among $\{variable, entity, type\}$, e.g. the mention and tag for the node $?class$ is "class" and variable, respectively. Further, we link all non-variable nodes to KB entries, e.g. the $type$ node with mention "person" links to *dbo:person* in Figure 1. Also, we decide the target ("select") node of the graph and add undirected edges between the nodes that are connected in the query graph, resulting in a graph structure representing the connectivity of all nodes.

Since all edges above are undirected and unlabeled, we fill in the exact KB predicate of each edge in the relation extraction stage (bottom-right in Figure 1) to construct a complete query graph.

Finally, we compose a SPARQL w.r.t. the query graph as output. Note that the body of the SPARQL exactly corresponds to the query graph, so only the SPARQL header is yet undetermined. Like Hu et al., 2021, we collect frequent trigger words in the train data to classify questions into COUNT, JUDGE or SELECT queries as in Table 1 (e.g. a question beginning with "is" triggers JUDGE). Thus, an entire SPARQL can now be formed. In the following sections, we expatiate our methodology for the two aforementioned stages.

3 Graph Structure Generation (GSG)

The overview of the model proposed for graph structure generation is illustrated by Figure 2. As discussed in Section 1, the model jointly deals with node extraction and graph composition via causal-modelling, which is detailed in this section below.

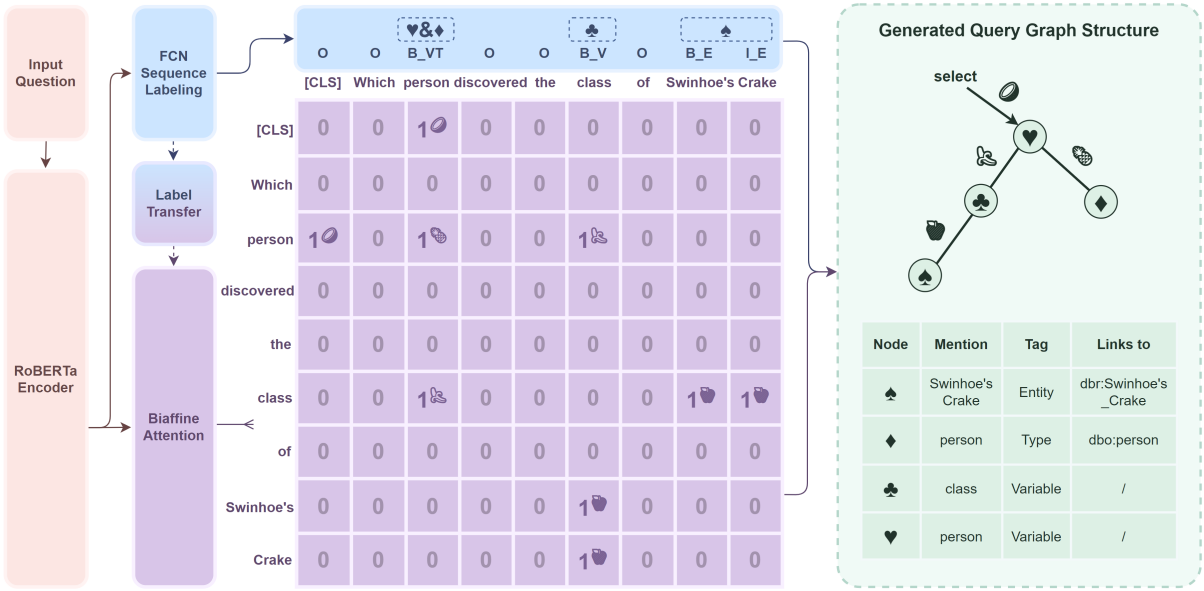

Figure 2: Causal-enhanced table-filling model for graph structure generation. The label-to-node and table-to-edge correspondence is illustrated by the poker and fruit symbols respectively.

3.1 Node Extraction (NE)

Node extraction discovers all nodes in the query graph, i.e. {*?person*, *?class*, *dbr:Swinhoe's_Crake*, *dbo:person*} in Figure 1. We represent a node by its mention and tag, i.e. ("person", *variable*), ("class", *variable*), ("Swinhoe's Crake", *entity*) and ("person", *type*) for each node respectively.

This goal can naturally be achieved by multi-class sequence labeling. More specifically, let $\mathbf{Q} \in \mathbb{N}^n$ be the question (token ids) with length n, we first encode it into hidden features $\mathbf{H}_{rb}$ by a RoBERTa (Liu et al., 2019) encoder $E_{rb} : \mathbb{N}^n \rightarrow \mathbb{R}^{n \times h_{rb}}$ with hidden size h_{rb}:

$$\mathbf{H}_{rb} = E_{rb}(\mathbf{Q}) \in \mathbb{R}^{n \times h_{rb}}$$

Then, $\mathbf{H}_{rb}$ is projected by a fully-connected-network (FCN) $E_{ne} : \mathbb{R}^{n \times h_{rb}} \rightarrow \mathbb{R}^{n \times |L|}$ into $\mathbf{Y}_{ne}$ in label space:

$$\mathbf{Y}_{ne} = E_{ne}(\mathbf{H}_{rb}) \in \mathbb{R}^{n \times |L|}$$

$L = \{O\} \cup \{B, I\} \times \{V, E, T, VT\}$ is the label set denoting the mention span of variables (V), entities (E), types (T), or overlapping variable and type (VT). Now, the label prediction of each token can be given by $\mathbf{P}_{ne} = argmax(\mathbf{Y}_{ne})$; also, given the gold token labels $\mathbf{G}_{ne} \in \mathbb{N}^n$ (Figure 2 top), a model for NE can be trained by optimizing:

$$\ell_{ne} = -\frac{1}{n} \sum_{i=1}^{n} log(\text{softmax}(\mathbf{Y}_{ne})[i; \mathbf{G}_{ne}[i]])$$

Where $[\cdot]$ denotes tensor indexing.

After detecting all node mentions and tags, we link each non-variable node to KB entries by DBpedia Lookup and a mention-to-type dictionary built on train data to align the graph structure with KB. See Appendix A for more details in node linking.

3.2 Graph Composition (GC)

After node extraction, all nodes in the query graph remain unconnected. To form the structure of the query graph, graph composition inserts unlabeled and undirected edges between the nodes that are related in the query graph, leaving the specific predicate of each edge yet unresolved. Formerly, graph composition is commonly modelled as a edge-sequence-generation process via stage-transition (Yih et al., 2015; Hu et al., 2018) or generative decoders (Shen et al., 2019; Chen et al., 2021). Despite the strong expressiveness, modelling graph composition by a sequence usually suffers from two issues: 1) while the edge sequence is ordered, edges in the query graph are a set without order. For a graph with two edges e_1 and e_2, both sequence e_1-e_2 and e_2-e_1 correctly represents the edges in the graph, but they are distinct from the perspective of sequence-generation. As a result, the edge set itself becomes ambiguous for the sequence, which confuses the model when comprehending a sequence and potentially decelerates the convergence. 2) As discussed by Zhang et al., 2019, without extra augmentation, sequence-generation

generally endures an exposure bias between training and inference, harming the model's accuracy when predicting. Hence, a robust model should address the issues above properly.

Here, we model graph composition by a table-filling process to decide all edges simultaneously involving no sequence-generation, which naturally circumvents all issues above. Let $\mathbf{H}_{gc} \in \mathbb{R}^{n \times h_{gc}}$ be the hidden features for graph composition (the full definition of $\mathbf{H}_{gc}$ with causal-modelling is given in Section 3.3; without causal-modelling, we simply have $\mathbf{H}_{gc} = \mathbf{H}_{rb}$), we adopt a biaffine attention model (Dozat and Manning, 2017; Wang et al., 2021) to convert $\mathbf{H}_{gc}$ into a table denoting the relationship between each token pair. More specifically, through two multi-layer-perceptrons (MLP) E_{head} and $E_{tail} : \mathbb{R}^{n \times h_{gc}} \rightarrow \mathbb{R}^{n \times h_{bi}}$, we first project $\mathbf{H}_{gc}$ into head ($\mathbf{H}_{head}$) and tail ($\mathbf{H}_{tail}$) features:

$$\mathbf{H}_{\{head,tail\}} = E_{\{head,tail\}}(\mathbf{H}_{gc}) \in \mathbb{R}^{n \times h_{bi}}$$

Then, for $\forall 1 \leq i, j \leq n$, the biaffine attention is performed between the head features of the i$^{\text{th}}$ token $\mathbf{h}^{(i)}_{head}$ and the tail features of the j$^{\text{th}}$ token $\mathbf{h}^{(j)}_{tail}$, producing $\mathbf{s}_{i,j} \in \mathbb{R}^2$ representing the probability that an edge exists between the i$^{\text{th}}$ and j$^{\text{th}}$ token:

$$\mathbf{s}_{i,j} = \text{softmax}(\text{Biaff}(\mathbf{h}^{(i)}_{head}, \mathbf{h}^{(j)}_{tail}))$$

$$\text{Biaff}(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T \mathbf{U}_1 \mathbf{y} + \mathbf{U}_2(\mathbf{x} \oplus \mathbf{y}) + \mathbf{b}$$

As $\mathbf{U}_1 \in \mathbb{R}^{2 \times h_{bi} \times h_{bi}}$, $\mathbf{U}_2 \in \mathbb{R}^{2 \times 2h_{bi}}$ and $\mathbf{b} \in \mathbb{R}^2$ are trainable parameters, $\oplus$ denotes concatenation. Combining all scores by $\mathbf{Y}_{gc} = (\mathbf{s}_{i,j})_{(1 \leq i,j \leq n)} \in \mathbb{R}^{n \times n \times 2}$, we now have a table describing the edge existence likelihood between any two tokens.

At training, we first obtain the boolean gold table $\mathbf{G}_{gc} \in \mathbb{B}^{n \times n}$, for every connected node pair in the query graph, the element in $\mathbf{G}_{gc}$ corresponding to any pair of tokens belonging to the mentions of the two nodes respectively is set to 1 (resulting in several rectangles of 1s). Also, we prefix the question with a special [CLS] token and connect it with the target node to represent the "select" edge; for ASK queries without target nodes, a [SEP] token is suffixed and connected with [CLS]. Note that since the graph structure is undirected, $\mathbf{G}_{gc}$ is a symmetric matrix. An example of $\mathbf{G}_{gc}$ can be found in Figure 2. With $\mathbf{G}_{gc}$, we can train the table-filler by ℓ_{tb}:

$$\ell_{tb} = -\frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} log(\mathbf{Y}_{gc}[i; j; \mathbf{G}_{gc}[i; j]])$$

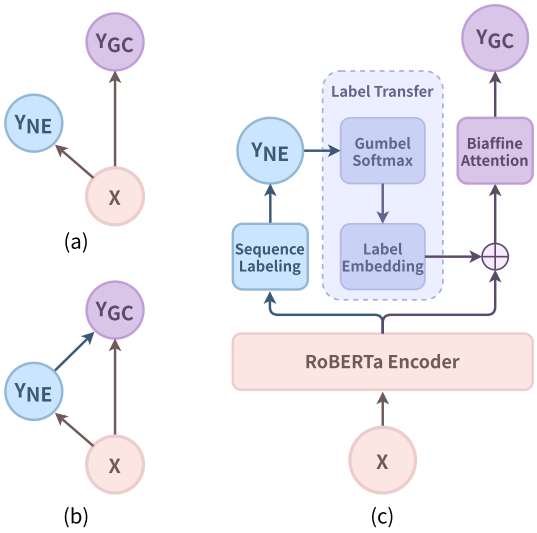

Figure 3: Modelling NE and GC with (a) and without (b) causality, as X, Y_{NE}, and Y_{GC} denotes question, NE predictions, and GC predictions. Model (c) learns the causal effects by a label transfer module.

Following Wang et al., 2021, we also introduce ℓ_{sym} to grasp the table symmetry. Finally, we optimize $\ell_{gc} = \ell_{tb} + \ell_{sym}$ to train a model for GC.

$$\ell_{sym} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{2} |\mathbf{Y}_{gc}[i; j; k] - \mathbf{Y}_{gc}[j; i; k]|$$

At inference, for each pair of nodes given by NE, we average the rectangle area in $\mathbf{Y}_{gc}$ corresponding to the mentions of the node pair as its edge existence probability. The node pairs with a probability higher than 0.5 are connected. This threshold is selected intuitively to denote an edge is more likely to exist against to not exist, though we argue that the prediction is insensitive to any threshold in reasonable range (e.g. 0.3~0.7).

3.3 Causal Modelling NE and GC

Up to now, NE and GC are treated as separate tasks that fail to model the intrinsic causal effects between them (e.g. edges in Y_{GC} only exist between the mentions detected in NE). Here, we model such causality by a mediation assumption in Figure 3(b) denoting the causal dependence of GC on both question and NE prediction by edge $\text{X} \rightarrow \text{Y}_{\text{GC}}$ and $\text{Y}_{\text{NE}} \rightarrow \text{Y}_{\text{GC}}$ respectively. To grasp this causal graph, we devise a label transfer (Chen et al., 2020a) module to enable the transfer of NE predictions to GC, i.e. representing $\text{Y}_{\text{NE}} \rightarrow \text{Y}_{\text{GC}}$, in Figure 3(c).

In detail, we sample NE predictions $\widetilde{\mathbf{Y}_{ne}}$ by gumbel softmax (Nie et al., 2019) with $\boldsymbol{g}$~Gumbel(0,1)

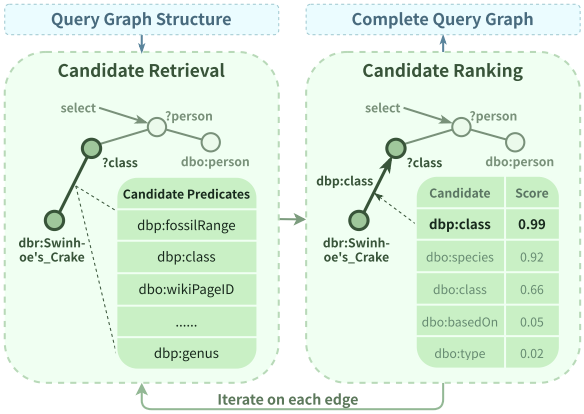

Figure 4: Candidate retrieval and raking framework for relation extraction.

and temperature τ.

$$\widetilde{\mathbf{Y}_{ne}} = \text{softmax}((\mathbf{Y}_{ne} + \boldsymbol{g})/\tau) \in \mathbb{R}^{n\times|L|}$$

$\widetilde{\mathbf{Y}_{ne}}$ is then embedded by label embedding $\mathbf{W}_{le} \in \mathbb{R}^{|L|\times h_{le}}$ and concatenated with $\mathbf{H}_{rb}$ to form $\mathbf{H}_{gc}$ in Section 3.2 with $h_{gc} = h_{rb} + h_{le}$:

$$\mathbf{H}_{gc} = \mathbf{H}_{rb} \oplus (\widetilde{\mathbf{Y}_{ne}}\mathbf{W}_{le}) \in \mathbb{R}^{n\times h_{gc}}$$

Now, by minimizing $\ell_{gsg} = \ell_{ne} + \ell_{gc}$, a joint model for NE and GC can be obtained. In this model, GC receives NE labels to learn the causal effects from NE, while NE gets feedback through differentiable label transfer to further aid GC decision. In this sense, our model improves the integrity of graph structure generation compared with separately modelling each subtask or simple multitasking.

4 Relation Extraction (RE)

As shown in Figure 4, relation extraction (RE) conducts candidate retrieval and ranking in turn for each edge in graph structure S to decide its predicate. For a question q, an edge e connecting nodes n_1 and n_2 with mention m_1,m_2 respectively, candidate retrieval recalls a set of predicates P that can be bound to e. Note that unlike e, each predicate in P is directional. Then, candidate ranking $\texttt{Rank}(P,q,m_1,m_2)$ gives each predicate a score. This section details this procedure.

Candidate Ranking For each $p_i \in P$, we encode it together with q,m_1,m_2 by a RoBERTa encoder and pool them to $0 \le s_i \le 1$ to score the predicate. If the direction of p_i is $n_1 \to n_2$, we join q, m_1, m_2, p_i sequentially by [SEP] token as model input; otherwise (direction $n_2 \to n_1$), the join order is q, m_2, m_1, p_i. By giving s_i to each candidate, we can get the most proper predicates for e by selecting those with highest scores. More details on training the ranking model can be found in Appendix B.

Algorithm 1: BeamSearchRE

Input: Question q, Query graph structure S, beam width b
Output: A beam of query graphs B

$B \leftarrow \{\{\}\}$;// Start with an empth graph
$S_{pend} \leftarrow S$;// All edges are pending
while $S_{pend} \neq \varnothing$ **do**
  $B' \leftarrow \{\}$;
  // Select a pending edge
  $e = (n_1, n_2) \leftarrow \texttt{Sample}(S_{pend})$;
  for $G \in B$ **do**
    $P \leftarrow \texttt{Retrieve}(G, n_1, n_2)$;
    // n_1/n_2 has mention m_1/m_2
    $C = \{(p_i, s_i)\} \leftarrow \texttt{Rank}(P, q, m_1, m_2)$;
    // Extend previous beams
    for $(p_i, s_i) \in C$ **do**
      $B' \leftarrow B' \cup \{G \cup \{(n_1, n_2, p_i, s_i)\}\}$;
  $B \leftarrow B'.\text{topk}(b)$;// Set up new beams
  // Mark e as determined
  $S_{pend} \leftarrow S_{pend} \setminus \{e\}$;

Candidate Retrieval Zhang et al., 2021 proposed a straightforward way to retrieve candidates: if either n_1 or n_2 is a non-variable node, the predicates around that node in KB are viewed as candidates; otherwise, they trace n_1 or n_2 in other graph edges with non-variable nodes and view the predicates k-hop away from that node in KB as candidates (e.g. predicates 2-hop away from *dbo:person* are candidates for $?class$-$?person$ in Figure 4). We view this as the baseline in latter experiments.

However, this results in a candidate scale $\text{O}(n^k)$[1], making it unscalable to multi-hop queries (k↑) and large KBs (n↑). Here, we propose Algorithm 1 to limit the scale to O(n). We start by selecting an edge between n_1^a and n_1^b containing a non-variable node (e.g. edge *?class-dbr:Swinhoe's_Crake* in Figure 4), retrieving all adjacent predicates of that node in KB and use $\texttt{Rank}$ to select the most proper predicate p_1 (e.g. dbp:named_by) of score s_1, this forms a subgraph G={(n_1^a,n_1^b,p_1)} with only one edge whose score is s_1. Then, we sample another edge between n_2^a and n_2^b (e.g. *?class-?person*) and retrieve its candidates P based on G (e.g. G already entails *?class=dbr:bird*, so all neighbors of *dbr:bird* forms P), this process is denoted as $\texttt{Retrieve}(G, n_2^a, n_2^b)$. Now, we use $\texttt{Rank}$ to se-

[1] n is the node degree in KB, k is the edge number in S

Type	Methods	P	R	F1
I	NSQA (Kapanipathi et al., 2021)	.448	.458	.445
	EDGQA (Hu et al., 2021)	.505	.560	.531
II	QAmp (Vakulenko et al., 2019)	.250	.500	.330
	NAMER (Zhang et al., 2021)	.438	.438	.435
	STaG-QA (Ravishankar et al., 2021)	**.745**	.548	.536
	Crake (ours)	.722	**.731**	**.715**

Table 2: End-to-end performance on LC-QuAD 1.0 test set. I/II stands for methods with/without aux tools. We re-implement NAMER since its results on LC-QuAD is not provided; however, NAMER suffers from severe timeout issues on DBpedia to limit its performance, so we restrict each candidate query to run at most 45s in practice (which already requires ~15h for a complete evaluation run).

lect p_2 of score s_2 from P, add (n_2^a,n_2^b,p_2) to subgraph G and update its score as $s_1 * s_2$. Repeating this loop until all edges are bound with a predicate, we finally form a query graph.

Note that for each edge, the candidate scale given by `Retrieve` is O(n), since it is always among the neighbors of one or several KB nodes. Also, to improve the recall of query graphs, this process can trivially be extended as a beam search with each step maintaining a beam of subgraphs B, ordering each subgraph by $\prod_i s_i$ as in Algorithm 1.

5 Experiments

Dataset We adopt LC-QuAD 1.0 (Trivedi et al., 2017), a predominant open-domain English KBQA benchmark based on DBpedia (Auer et al., 2007) 2016-04, to test the performance of our system. We randomly sample 200 questions from train data as dev set and follow the raw test set, resulting in a 4800/200/1000 train/dev/test split. More details on the dataset can be found in Appendix C. Like Zhang et al., 2021, we do not experiment on multiple datasets due to the high annotation cost involved, however, we conduct no dataset-specific optimizations in this work, so we consider the large improvements on LC-QuAD and detailed discussions sufficient to prove our effectiveness.

Annotation We annotate the dataset with the mention of each node in query graph, e.g. the mention "class" and "person" for the node *?class* and *dbo:person* respectively in Figure 1. With the annotation, we obtain the gold data (G_{ne},G_{gc}) to train our models. Appendix D details the annotation process.

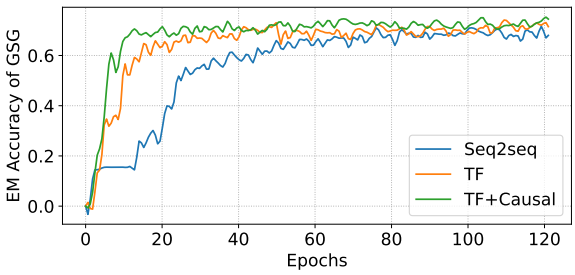

Figure 5: EM accuracy of GSG during training. See the meaning of each series in Table 3.

Baselines We evaluate our method against existing works both with and without auxiliary tools. With aux tools, Kapanipathi et al., 2021 constructs query graphs based on the AMR of questions; Hu et al., 2021 designs rules on constituency tree to aid query graph formation. For independent pipelines without aux tools, Vakulenko et al., 2019 parses URI mentions from the question to match with KB via confidence score passing; Ravishankar et al., 2021 combines a generative graph-skeleton decoder with entity and relation detector to form a query; Zhang et al., 2021 co-trains a pointer generator with the node extractor to build a query graph, it's worth to note that this work also requires the node-to-mention Annotation for training.

Setup We utilize the RoBERTa-large released by huggingface (Wolf et al., 2020) as our encoder. All experiments are averaged on two runs on an NVIDIA A40 GPU. For the GSG model, we train for at most 500 epochs (~6 GPU-hours) and report the best checkpoint on dev set; for the RE model, we train for 20 epochs (~16 GPU-hours) and report the final checkpoint. For hyperparameters, we find no apparent performance variance on dev set as long as the values are in reasonable range (e.g. $64 \le h_{le} \le 1024$, $1e\text{-}6 \le lr_{gsg} \le 2e\text{-}5$) so no further tuning is involved. See the full setting in Appendix E.

5.1 End-to-end Evaluation

As shown in Table 2, our method, Crake, outperforms all former methods by a large ~17% margin on F1, becoming the new SoTA of LC-QuAD 1.0. Surpassing methods requiring aux tools (I) on all metrics, we present the effectiveness of independent pipelines (II) that avoid cascading errors. Also, we achieve consistent answer precision and recall to surpass other methods in II on F1, showing the superiority of our pipeline design, which is further discussed in the sections below.

Methods	Decoder Parameters	NE Accuracy			GSG Accuracy		End-to-end		
		P	R	F1	EM	Actual	P	R	F1
Seq2seq	76.67M (×1)	.895	.901	.897	.695	.768	.653	.674	.654
TF	0.66M (×1/100)	.895	.901	.897	.728	.795	.655	.674	.657
TF+SMTL	0.66M (×1/100)	.901	.904	.902	.735	.805	.665	.684	.667
TF+Causal	3.03M (×1/25)	**.909**	**.914**	**.911**	**.755**	**.828**	**.677**	**.696**	**.680**

Table 3: Experiments on table-filling and causal-modelling. Seq2seq and TF adopt a generative decoder and table-filler in GC respectively, while both deal with NE and GC by separate models. TF+SMTL (simple multitask learning) co-trains NE and GC by directly adding losses without modelling their intrinsic causal effects. TF+Causal denotes our full approach which models the causal effects between NE and GC by label transfer. We report the node-level P/R/F1 in NE, the exact-match (EM) and actual accuracy (that ignores variable mentions in judging accuracy) in GSG, and the overall answer-level P/R/F1 on LC-QuAD 1.0 dev set for comparison.

5.2 Effects of Tabel-Filling

As explained in Section 3.2, modelling GC as a sequence-generation causes a few issues that can be overcome by table-filling. Specifically, the sequence ambiguity confuses the learning process and requires large decoders to grasp the sequence generation policy, which may slow down the convergence. Besides, the exposure bias harms the decoding accuracy of the model at inference. This section, we try to verify such effects by experiments. To enable the comparison with sequence-generation, we construct a generative decoder as in Zhang et al., 2021 as the baseline, which sequentially generates the connected node pairs in the graph structure to represent the edges. We train the generative model under the same settings (e.g. learning rate, warmup, epochs, etc.), resulting in the performance of `Seq2seq` in Table 3.

Comparing with the table-filling model (i.e. `TF` in Table 3), `Seq2seq` comes short in the accuracy of graph structure, indicating the negative effects of the exposure bias on predicting accuracy. Meanwhile, `TF` requires only 1/100 of `Seq2seq`'s parameters to achieve comparable or better results, we attribute this to the removal of sequence ambiguity which frees the model from acquiring the complex and ambiguous scheme of sequence-generation. This speculation is further verified in Figure 5, in which `TF` converges distinctly quicker than `Seq2seq` since the simultaneous decision of all edges is well-defined and easier to learn. Thus, compared with sequence-modelling, handling GC via table-filling reduces model size and boosts training, which is essential for real-world applications.

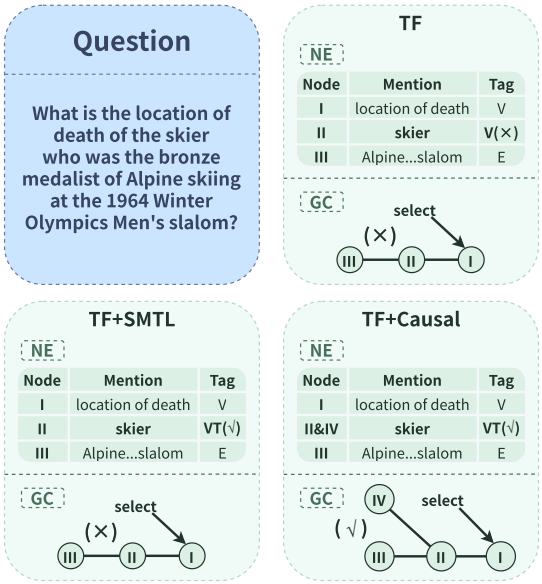

Figure 6: Case study on the effects of causal-modelling.

5.3 Effects of Causal-Modelling

We propose a joint model to learn the NE-GC causalities in Section 3.3, to discuss its effects, we compare it with two alternatives in Table 3: 1) using two separate models for NE and GC (`TF` in Table 3) like Ravishankar et al., 2021, 2) co-training NE and GC by sharing encoder and adding losses (`TF+SMTL` in Table 3) like Shen et al., 2019. As shown, co-training consistently surpasses separate models by grasping the shared knowledge between NE and GC, nevertheless, our causal-modelling approach (`TF+Causal`) further outperforms co-training. In detail, though `TF+Causal` has similar results with `TF+SMTL` in NE, it achieves better accuracy for overall GSG (NE+GC) and excels in end-to-end metrics. Therefore, we infer that causal-modelling improves the integrity of the GSG stage

Methods	Accuracy			Efficiency		
	P	R	F1	1-hop	2-hop	3-hop
Baseline	.560	.566	.556	**0.12s**	42.4s	84.2s
BeamSearch	**.677**	**.696**	**.680**	**0.12s**	**1.06s**	**2.72s**

Table 4: Performance comparison between our beam-search RE algorithm and its baseline in Section 4. Accuracy refers to the answer-level P/R/F1, efficiency is measured by the average run time on 1/2/3-hop queries.

by expressing the internal causalities between its subtasks. To better understand this, we perform a case study in Figure 6, in which `TF` fails to realize that "skier" also corresponds to a type node; in contrast, `TF+SMTL` extract all nodes correctly by learning both NE and GC labels, but it still fails in generating a correct graph structure. Finally, `TF+Causal` utilizes the VT tag of "skier" in NE predictions and correctly connects the II-IV edge in GC. Thus, Figure 6 demonstrates the usage of causal effects to reach higher accuracy in GSG.

5.4 Analysis on Beam-Search RE

In this section, we compare our beam-search RE algorithm with its baseline. As stated in Section 4, by alternately performing retrieval and ranking on each edge (rather than retrieving the candidates of every edge before ranking), our approach lowers the scale of candidate predicates on multi-hop queries to get better efficiency, which is verified in Table 4. In detail, `BeamSearch` costs substantially less time than `Baseline` in 2 and 3-hop queries (note that for 1-hop queries, two methods reduce to a same process with similar time costs). Since `BeamSearch` only operates on the neighbors of certain KB nodes, it avoids the retrieval of 2-hop neighbors, which requires considerable time on DBpedia, to improve efficiency. In addition, by pruning off useless candidates in `Baseline`, `BeamSearch` also achieves higher overall KBQA accuracy in Table 4. Therefore, Algorithm 1 transcends previous methods to reveal an efficient and accurate solution for ranking-based RE scalable to KB size and query complexity.

6 Related Works

KBQA via Semantic Parsing A mainstream to solve KBQA is semantic parsing (Yih et al., 2016) which converts a question to a KB query to get answers. Due to the graph-like structure of KB queries, prior works construct query graphs to represent queries in semantic parsing. Among them, some works (Zafar et al., 2018; Chen et al., 2020b) only focus on predicting the graph structure given node inputs. To perform end-to-end QA, Hu et al., 2017 leverages the dependency parsing tree to match KB subgraphs for answers; Kapanipathi et al., 2021 builds the query graph by transforming and linking the AMR (Banarescu et al., 2012) of the question; Hu et al., 2021 uses the constituency tree to compose an entity description graph representing the query graph structure. Requiring aux tools or data structures, these works may be subjected to cascading errors. Yih et al., 2015 overcomes this by an independent stage-transition framework to generate the query graph, Hu et al., 2018 extends the transitions to express more complex graphs. Besides, Zhang et al., 2021 adopts a pointer generator to decode graph structure, Ravishankar et al., 2021 generates the query skeleton by a seq2seq decoder. Unlike these methods that model the query graph as a sequence (by state-transition or generative decoder), we decode all edges at once via a table-filler in graph structure generation.

Modelling causal effects Causality occurs in various deep-learning scenarios between multiple channels or subtasks, existing works models the causality for better performance. Niu et al., 2021 mitigates the false causal effects in VQA (Antol et al., 2015) to overcome language bias; Zeng et al., 2020 dispels the incorrect causalities from different input channels of NER by generating counterfacts. Chen et al., 2020a utilizes the inter-subtask causalities to improve multitask learning for JERE (Li and Ji, 2014), ABSA (Kirange et al., 2014), and LJP. Unlike them, we formulate and utilize the internal causal effects in KBQA.

7 Conclusion

In this work, we formalize the generation of query graphs in KBQA by two stages, namely graph structure generation (GSG) and relation extraction (RE). In GSG, we propose a table-filling model for graph composition to avoid the ambiguity and bias of sequence-modelling, meanwhile, we encode the inherent causal effects among GSG by a label-transfer block to improve the stage integrity. In RE, we introduce an effective beam-search algorithm to retrieve and rank predicates in order for each edge, which turns out to be scalable for large KBs and multi-hop queries. Consequently, our approach substantially surpasses previous state-of-the-arts in

KBQA, revealing the effectiveness of our pipeline design. Detailed experiments also validate the effects of all our contributions.

8 Limitation

Admittedly, our approach endures certain limitations as discussed below.

Query Expressiveness Like most semantic parsing systems, we fail to cover all the operations of SPARQL, limiting our capability to compose queries with complex `filter` or property path. For the conciseness of our system, we only focus on constructing triples in the multi-hop query graph in this paper, while we plan to incorporate more functions into Crake in the future to improve the expressiveness of the system.

Annotation Cost Training models with node mentions require expensive manual annotations, which is impractical for us to conduct on every popular KBQA dataset. As explained in Section 5, without data-oriented optimization, we believe the significant gain presented adequate to verify our contributions. Further, we expect to extenuate such costs in two directions for the future: 1) some modules of our framework (e.g. NE) is generalizable to other English questions, gifting it the potential to be transferred to other datasets without re-training; 2) few-shot (Wang et al., 2020) and active (Aggarwal et al., 2014) learning techniques aids the model to reach competitive performance with a small portion of annotated data, which can be explored in our framework to reduce annotation cost.

Acknowledgements

This work was supported by National Key R&D Program of China (2020AAA0105200) and NSFC under grant U20A20174. The corresponding author of this work is Lei Zou (zoulei@pku.edu.cn). We would like to thank Zhen Niu and Sen Hu for their kind assistance on this work. We also appreciate anonymous reviewers for their valuable comments and advises.

References

Charu C Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and S Yu Philip. 2014. Active learning: A survey. In *Data Classification*, pages 599–634. Chapman and Hall/CRC.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: visual question answering. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2425–2433. IEEE Computer Society.

Ram G Athreya, Srividya K Bansal, Axel-Cyrille Ngonga Ngomo, and Ricardo Usbeck. 2021. Template-based question answering using recursive neural networks. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 195–198. IEEE.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2012. Abstract meaning representation (amr) 1.0 specification. In *Parsing on Freebase from Question-Answer Pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle: ACL*, pages 1533–1544.

Wenqing Chen, Jidong Tian, Liqiang Xiao, Hao He, and Yaohui Jin. 2020a. Exploring logically dependent multi-task learning with causal inference. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2213–2225, Online. Association for Computational Linguistics.

Yongrui Chen, Huiying Li, Yuncheng Hua, and Guilin Qi. 2020b. Formal query building with query structure prediction for complex question answering over knowledge base. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3751–3758. ijcai.org.

Yongrui Chen, Huiying Li, Guilin Qi, Tianxing Wu, and Tenggou Wang. 2021. Outlining and filling: Hierarchical query graph generation for answering complex questions over knowledge graph. *arXiv preprint arXiv:2111.00732*.

Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. 2017. Kbqa: Learning question answering over qa corpora and knowledge bases. *Proceedings of the VLDB Endowment*, 10(5).

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2017. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837.

Sen Hu, Lei Zou, and Xinbo Zhang. 2018. A state-transition framework to answer complex questions over knowledge base. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2098–2108, Brussels, Belgium. Association for Computational Linguistics.

Xixin Hu, Yiheng Shu, Xiang Huang, and Yuzhong Qu. 2021. Edg-based question decomposition for complex question answering over knowledge bases. In *International Semantic Web Conference*, pages 128–145. Springer.

Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramón Fernandez Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue-Nkoutche, et al. 2021. Leveraging abstract meaning representation for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3884–3894.

DK Kirange, Ratnadeep R Deshmukh, and MDK Kirange. 2014. Aspect based sentiment analysis semeval-2014 task 4. *Asian Journal of Computer Science and Information Technology (AJCSIT) Vol*, 4.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 402–412, Baltimore, Maryland. Association for Computational Linguistics.

Yinnian Lin, Minhao Zhang, Ruoyu Zhang, and Lei Zou. 2021. Deep-ganswer: A knowledge based question answering system. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 434–439. Springer.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Weili Nie, Nina Narodytska, and Ankit Patel. 2019. Relgan: Relational generative adversarial networks for text generation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yulei Niu, Kaihua Tang, Hanwang Zhang, Zhiwu Lu, Xian-Sheng Hua, and Ji-Rong Wen. 2021. Counterfactual vqa: A cause-effect look at language bias. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12700–12710.

Eric Prud'hommeaux. 2008. Sparql query language for rdf, w3c recommendation. *http://www. w3. org/TR/rdf-sparql-query/*.

Srinivas Ravishankar, June Thai, Ibrahim Abdelaziz, Nandana Mihidukulasooriya, Tahira Naseem, Pavan Kapanipathi, Gaetano Rossilleo, and Achille Fokoue. 2021. A two-stage approach towards generalization in knowledge base question answering. *arXiv preprint arXiv:2111.05825*.

Tao Shen, Xiubo Geng, Tao Qin, Daya Guo, Duyu Tang, Nan Duan, Guodong Long, and Daxin Jiang. 2019. Multi-task learning for conversational question answering over a large-scale knowledge base. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2442–2451, Hong Kong, China. Association for Computational Linguistics.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218. Springer.

Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 639–648. ACM.

Svitlana Vakulenko, Javier David Fernandez Garcia, Axel Polleres, Maarten de Rijke, and Michael Cochez. 2019. Message passing for complex question answering over knowledge graphs. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 1431–1440. ACM.

Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34.

Yijun Wang, Changzhi Sun, Yuanbin Wu, Hao Zhou, Lei Li, and Junchi Yan. 2021. Unire: A unified label space for entity relation extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 220–231.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390, Jeju Island, Korea. Association for Computational Linguistics.

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 *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)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

Hamid Zafar, Giulio Napolitano, and Jens Lehmann. 2018. Formal query generation for question answering over knowledge bases. In *European semantic web conference*, pages 714–728. Springer.

Xiangji Zeng, Yunliang Li, Yuchen Zhai, and Yin Zhang. 2020. Counterfactual generator: A weakly-supervised method for named entity recognition. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7270–7280, Online. Association for Computational Linguistics.

Minhao Zhang, Ruoyu Zhang, Lei Zou, Yinnian Lin, and Sen Hu. 2021. NAMER: A node-based multitasking framework for multi-hop knowledge base question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*, pages 18–25, Online. Association for Computational Linguistics.

Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. 2019. Bridging the gap between training and inference for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4334–4343, Florence, Italy. Association for Computational Linguistics.

A Details in Entity and Type Linking

We link each non-variable node to a KB entry by its mention. For entity nodes, we directly link it to an entity with the same name as its mention if such entity exists in the KB (e.g. link mention "New York" to dbr:New_York); otherwise, we recall entities by DBpedia Lookup[2] and further prioritize ones whose lower-cased name is the same as the lower-cased mention (e.g.dbr:new_york). Then, the prioritized entity with the highest lookup score is linked to the node; if no entity is prioritized, the entity with the highest score is selected.

For type nodes, we build a dictionary D based on the mention-type pairs (e.g. authors-dbo:Writer) in train data and directly use the link result from D if the mention exists in D. Otherwise, we singularize and capitalize the mention to construct an URI with prefix *dbo* (e.g. bands→dbo:Band), if this URI presents in the KB, the type node is linked to this URI. If no entry is found for either an entity or a type afterall, we simply discard the node from our query graph.

Note that although we involve no extra disambiguation step, DBpedia Lookup itself has certain mention-level disambiguation abilities to refine mention-relevant candidates. Admittedly, sentence context also contributes to a precise linking decision, leaving such context-level disambiguation a future direction to improve our work.

B Training Details of the Candidate Ranking Model

We mainly follow NAMER (Zhang et al., 2021) in training the candidate ranking model mentioned in Section 4. Basically, the positive and negative training samples are obtained from the gold query. For instance, in Figure 1, we obtain the candidates between ?class (m_1="class") and ?person (m_2="person") by constructing *"select ?r { ?person dbp:type dbo:person. ?class ?r ?person. dbr:Swinhoe's_Crake dbp:class ?class }"* and *"select ?r { ?person dbp:type dbo:person. ?person ?r ?class. dbr:Swinhoe's_Crake dbp:class ?class }"* with query results P_p and P_r respectively. Let p^*= $dbp:named\_by$ be the correct predicate, we collect model inputs $\{(q, m_1, m_2, p^*)\}$ as a positive sample (i.e. of label 1) and $\{(q, m_1, m_2, p_i)|p_i \in P_{pos} \backslash \{p^*\} \cup \{(q, m_2, m_1, p_i)|p_i \in P_{rev}\}$ as negative samples (i.e. of gold label 0).

Further, we follow the augmentation process in NAMER to learn the effects of mention order on model predictions. Specifically, we add $\{(q, m_1, m_2, p_i)|p_i \in P_{rev} \backslash \{p^*\} \cup \{(q, m_2, m_1, p_i)|p_i \in P_{pos}\}$ to negative samples when training. With the aforesaid process repeated

[2] http://wiki.dbpedia.org/projects/dbpedia-lookup

on each query graph edge, we get the full training samples to train a ranking model.

Besides, similar with NAMER, we observe a performance decay when forcibly co-training RE and GSG module, in this regard, we leave RE a separate module alongside GSG in the system. As discussed in NAMER, the different input channels between RE and GSG may result in unequal semantic spaces for the model. Thus, despite the causal association between RE and GSG, we conjecture that the model fails to acquire beneficial causalities between incompatible semantic spaces.

C Details of the Dataset

LC-QuAD 1.0 is an English open-domain KBQA dataset widely used to evaluate KBQA systems. With a GPL-3.0 licence, this dataset is intended for training and testing models to answer a question via querying the knowledge base, permitting modifications on the dataset for experiments, which is consistent with the way we use the dataset (annotating node mentions for each data entry, train several models for KBQA on the train data and test the system performance on the test data).

Due to the nature of KBQA tasks, LC-QuAD 1.0 involves questions about certain real-world entities usually including persons, organizations or objects (which is exactly the conditions where KBQA is applied to real-world applications). However, most information about the individuals (e.g. name, team, etc.) are publicly available (since the dataset utilizes DBpedia as background KB while DBpedia mainly collects data from publicly available Wikipedia). Further, when annotating the dataset, we perform a brief manual check on potential offensive or biased contents, to the best of our efforts, we find no apparent offensive hints in the questions and SPARQL queries. Hence, we believe that LC-QuAD 1.0 under intended KBQA use has minor potential to offend others or cause privacy issues.

D Details of Data Annotation

Annotation Guidelines We adopt the same annotation format as Zhang et al., 2021 to annotate the LC-QuAD 1.0 dataset. Specifically, for each node in the query graph corresponding to the SPARQLs in the dataset, the mention of such node in the question is annotated. All annotated mentions are required as whole-words (e.g. including the 's' for plural words), the mention is left as "None" when no mention of a node can be found. There are certain cases where multiple mentions co-refer a node, we encourage annotators to choose a mention containing more concrete semantics, while all of these mentions are acceptable (e.g. for the question "Who is Jack's dad?", both "Who" and "dad" are correct mentions but the latter is encouraged since it indicates more semantics of the node). We provide a detailed guideline[3] to annotators with extra discussions on marginal cases to further aid the annotation. Also, we discuss the potential risks and the overall usage of such annotations to get agreements from the annotators in the guideline.

Annotation Process We recruit 9 annotators with necessary background knowledge from school, consisting of 5 undergraduate and 4 graduate students, to fulfil the annotation task. By completing the annotation, we provide essential payments for each annotator. Finally, we use a script to auto-check the collected annotations and perform basic corrections (e.g. align all mentions to whole-words).

E Hyperparameter Settings

Table 5 details our hyperparameter settings.

Name	Description	Setting
h_{rb}	Hidden size of the RoBERTa encoder	1024
h_{bi}	Hidden size of the biaffine model	256
h_{le}	Dimension of the label embedding	256
τ	Gumbel-softmax temperature	0.05
$optim$	Optimizer to train both GSG and RE models	AdamW
β_1/β_2	Betas of the AdamW optimizer	0.9 / 0.9
wd	Weight decay rate of the AdamW optimizer	1e-5
lr_{rb}	Learning rate of the RoBERTa encoder in GSG	1e-5
lr_{gsg}	Learning rate of other parameters in GSG	5e-5
$batch_{gsg}$	Batch size of the GSG model	64
lr_{re}	Learning rate of the RE model	1e-5
$batch_{re}$	Batch size of the RE model	100
b	Beam width in RE	4

Table 5: Detailed hyperparameter settings in this work.

F Ethical Statements

Considering the nature of NLP-based QA systems, our method keeps the risk to output false (e.g. incorrect answers to factoid questions) or biased (e.g. imprecise count of answer numbers) answers, which might cause issues in trustworthy or practical uses. However, we'd like to clarify that this work is intended for discovering more accurate and efficient systems on KBQA regardless of the exact content in a KB, the answers to specific questions given by our method does not reflect the authors' point of view.

[3] See the guideline in supplementary materials.