AutoQGS: Auto-Prompt for Low-Resource Knowledge-based Question Generation from SPARQL

Guanming Xiong
Peking University
Beijing, China
gm_xiong@pku.edu.cn

Junwei Bao
JD AI Research
Beijing, China
baojunwe001@gmail.com

Wen Zhao
Peking University
Beijing, China
zhaowen@pku.edu.cn

Youzheng Wu
JD AI Research
Beijing, China
wuyouzheng1@jd.com

Xiaodong He
JD AI Research
Beijing, China
hexiaodong@jd.com

ABSTRACT
This study investigates the task of knowledge-based question generation (KBQG). Conventional KBQG works generated questions from fact triples in the knowledge graph, which could not express complex operations like aggregation and comparison in SPARQL. Moreover, due to the costly annotation of large-scale SPARQL-question pairs, KBQG from SPARQL under low-resource scenarios urgently needs to be explored. Recently, since the generative pre-trained language models (PLMs) typically trained in natural language (NL)-to-NL paradigm have been proven effective for low-resource generation, e.g., T5 and BART, how to effectively utilize them to generate NL-question from non-NL SPARQL is challenging. To address these challenges, AutoQGS, an auto-prompt approach for low-resource KBQG from SPARQL, is proposed. Firstly, we put forward to generate questions directly from SPARQL for the KBQG task to handle complex operations. Secondly, we propose an auto-prompter trained on large-scale unsupervised data to rephrase SPARQL into NL description, smoothing the low-resource transformation from non-NL SPARQL to NL question with PLMs. Experimental results on the WebQuestionsSP, ComlexWebQuestions 1.1, and PathQuestions show that our model achieves state-of-the-art performance, especially in low-resource settings. Furthermore, a corpus of 330k factoid complex question–SPARQL pairs is generated for further KBQG research. 1

CCS CONCEPTS
• Computing methodologies → Natural language generation.

KEYWORDS
Knowledge Graph, Complex Question Generation, Low Resource

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Guanming Xiong, Junwei Bao, Wen Zhao, Youzheng Wu, and Xiaodong He. 2022. AutoQGS: Auto-Prompt for Low-Resource Knowledge-based Question Generation from SPARQL. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557246

1 INTRODUCTION
Knowledge-Based Question Generation (KBQG) is a task that aims to generate natural language questions given a knowledge base (KB). KBQG has a wide range of applications and has won wide attention in academia and industry. In this paper, we use a specific knowledge graph (KG), Freebase [4], as KB. Recent works mainly adopt sequence-to-sequence neural networks to generate questions given a Resource Description Framework [15, 16] (RDF) graph that is a directed graph composed of triple statements in a knowledge graph. For a single relation graph, a series of works [3, 8, 14] enriched the input with auxiliary information, equipped the decoder with copy mechanisms, and devised delicate models to improve the fluency or the semantic accuracy of generated questions. For star-like graph, [11] used Transformer encoder while [5] applied a bidirectional Graph2Seq model to encode the graph structure. Very recently, JointGT [9] adopted the modern pre-trained language model BART [12] to generate questions, achieving state-of-the-art performance.

However, KBQG still faces three major challenges:

1. Absence of complex operations. The existing KBQG methods mainly generated questions from an RDF graph. However, RDF is a standard model for data interchange on the Web, it’s oriented to describe resources not to express constraints on top of them, such as aggregation, superlative, and comparative questions. Compared with the RDF graph, SPARQL has a complete semantic representation that covers all the question types mentioned above. How to generate questions based on SPARQL is a challenge.

2. Low resources. For expressing the predicates in KG to NL form, traditional supervised methods tended to annotate large-scale SPARQL–question pairs. However, the labor cost is enormous. Besides, KG may contain complex schemas, such as Compound Value.
Type\(^2\) (CVT) in Freebase. Each CVT combination has a different meaning, hence it is difficult for annotations to cover all combinations. How to generate questions without sufficient resources is still under-explored to date.

3. Efficient generation. The advanced generative pre-trained language models (PLMs) have been proven effective in natural language generation (NLG) tasks \([6, 9, 12]\). Nonetheless, PLMs were trained in NL-to-NL paradigm, but the SPARQL expression is different from the NL form. Therefore, how to leverage the strengths of PLMs to generate high-quality questions matters a lot.

To address the challenges mentioned above, we propose AutoQGS, an auto-prompt approach for low-resource KBQG from SPARQL. Figure 1 shows the overall process. Firstly, we incorporate SPARQL expression directly as input, which retains the original semantics. Secondly, we propose a model, auto-prompter, to rephrase the SPARQL to the corresponding NL description, named prompt text. Auto-prompter combines the strengths of distant supervision \([17]\) and the strong generative ability of PLMs. Specifically, the training data of auto-prompter are subgraphs that could be massively sampled from KB, and the target prompt text is collected from large-scale corpus as well. Lastly, we explore an efficient question generation method in low-resource scenarios. Our model significantly outperforms existing state-of-the-art baselines by a large margin, especially in low-resource settings.

The main contributions of this paper are summarized as follows.

- Put forward to generate questions directly from SPARQL for KBQG task to handle complex operations.
- Propose AutoQGS, an approach to rephrase RDF graph to prompt text, and generate questions from SPARQL and prompt text.
- Conduct extensive experiments on two datasets, and the results show that AutoQGS improve the performance observably.

---

\(^2\)A Compound Value Type is a Type within Freebase which is used to represent data where each entry consists of multiple fields. See https://developers.google.com/freebase/guide/basic_concepts

2 RELATED WORK

KBQG has come a long way in the past decades. In the early time, many works generate questions in template-based approaches. \([26]\) proposed an unsupervised system to collect questions by a search engine from a small number of template-based questions. \([24, 25]\) collect structured triple-pattern query from seed question and use a template-based method to verbalize the structured query. However, template-based approaches lack flexibility, and the annotation cost is high.

Recent works for KBQG are mainly based on sequence-to-sequence neural networks given a set of subgraphs from knowledge graph (KG). \([23]\) first used a neural network for encoding KG fact triples into natural language questions and generated the 30M Factoid Question-Answer datasets. However, it was trained on a mass of fact-question pairs, which is challenging to collect. \([21]\) proposed an RNN-based model to generate simple questions and corresponding answers by converting all the KG entities to a set of keywords. \([1]\) and \([2]\) developed a flexible copying mechanism to alleviate the rare words problem. For unseen predicates and entity types problem, \([8]\) collected textual contexts in the Wikipedia as auxiliary information and adopted a part-of-speech copy action mechanism to generate questions. However, \([14]\) thought the textual contexts were noisy or even wrong. They presented a complicated model that integrates diversified off-the-shelf contexts and devised an answer-aware loss to make sure the generated questions are associated with a definitive answer. Based on the Transformer \([28]\), \([11]\) proposed an end-to-end neural network-based method for generating complex multi-hop and difficulty-controllable questions over a subgraph in KG. \([3]\) focus on semantic drift problem. They proposed to incorporate auxiliary information and word types in generated questions, make the decoder output conditioned on these types, and design a DPT-based evaluator to encourage question structural conformity in a reinforcement learning framework. Instead of using a set of KG triples, \([5]\) proposed to apply a bidirectional Graph2Seq model to encode the KG subgraph and target answers, and then generate questions with a node-level copying mechanism. \([9]\) proposed a pre-trained model called JointGT for KG-to-text generation tasks. They added a structure-aware semantic aggregation module in BART.
to model the structure of input graphs. They then pre-trained the model in a large-scale corpus, and fine-tuned in question generation task.

In comparison, our approach utilizes advanced generative pre-trained language models to generate questions from SPARQL, rather than either make templates to rephrase questions or generate questions from an KG subgraph.

3 APPROACH

3.1 Problem Formulation

This study investigates the task of knowledge-based question generation (KBQG). Conventional KBQG works generated questions from fact triples in the knowledge graph, which could not express complex operations. A SPARQL expression is of complete syntax, which is capable of fully formalizing questions with complex operations\(^3\), e.g., aggregation(COUNT), comparative(<,>,<=,>=), and superlative(OPTION BY ?x LIMIT 1). Moreover, due to the costly annotation of large-scale SPARQL-question pairs, KBQG from SPARQL under low-resource scenarios urgently needs to be explored. In this paper, we propose to generate question directly from SPARQL for low-resource KBQG task. Given an executable SPARQL expression \( S \) and a knowledge graph \( K \), our goal is to generate a natural language (NL) question \( Q \) that is consistent with the SPARQL. Given the above definitions, the task can be formalized as learning the distribution \( p(Q|S,K) \).

3.2 Model Overview

Recently, the generative pre-trained language models (PLMs) typically trained in NL-to-NL paradigm have been proven effective for low-resource generation, e.g., T5\(^6\) and BART\(^12\). However, generating questions directly from non-NL SPARQL is not friendly to the generative PLMs. In this paper, we propose AutoQGS, an auto-prompt approach which rephrases SPARQL to NL text automatically, smoothing the transformation from non-NL SPARQL to NL question. The overall process of AutoQGS is shown in Figure 1. Specifically, AutoQGS consists of two procedures, (1) auto-prompt from SPARQL to NL text, and (2) question generation (QG) based on SPARQL and NL prompt text.

Formally, auto-prompt aims to generate prompt text \( T \) from SPARQL \( S \). The process of auto-prompt can be formalized as

\[
T = \text{AP}(S; K; \Theta)
\]

(1)

Afterwards, \( S \) and \( T \) are concatenated as input\(^4\) of a question generator to produce a question \( \hat{Q} \). The procedure of question generation can be formalized as

\[
\hat{Q} = \text{QG}(S, T; \Phi)
\]

(2)

We will introduce the auto-prompt \( \text{AP}(\cdot) \) and the question generation \( \text{QG}(\cdot) \) procedures in Section 3.3 and Section 3.4, respectively.

\(^3\)Complex operations are defined as functions beyond the KG predicates.

\(^4\)We replace the machine identifiers of the topic entities in SPARQL with their surface names, but for brevity, we still use the symbol \( S \).

\(^5\)For brevity, we omit the prefix: PREFIX ns: <http://rdf.freebase.com/ns/>.

3.3 Auto-prompt

Auto-prompt \( \text{AP}(\cdot) \) generates prompt text \( T \) from SPARQL \( S \). Figure 2 shows the overview of auto-prompt. Given a SPARQL \( S \), we first execute it based on \( K \) and obtain an instantiated subgraph \( G \) via subgraph sampling defined in Section 3.3.1. \( G \) can be decomposed into a set of atomic subgraphs. Each \( g \in G \) is serialized as input of an auto-prompter and converted into text. They are then concatenated to construct the prompt text \( T \). Specifically, this procedure is formalized as follows:

\[
T = \text{CONCAT}(\text{BeamSearchTop1}(p_\Theta(d_i|d_{\text{out}}, g)|g \in G))
\]

(3)

\[
G = g(f(S, K))
\]

(4)

where function \( \text{CONCAT}(\cdot) \) means concatenating all the elements, \( \text{BeamSearchTop1} (\cdot) \) is the generation process with beam search (we keep the one with the highest score), \( \Theta \) is the learnable parameters of auto-prompter, \( d_i \) is the \( i \)-th predicted token, and function \( f(\cdot) \) indicates the graph sampling process. In this work, the auto-prompter supports both star and chain topologies, and the prompt text will vary as the input entities and their information change. By contrast, previous works only save the hard matched results for single-relation predicate (e.g. “is birthplace of” for “person/place_of_birth”).

3.3.1 Subgraph Sampling. Subgraph sampling is a process that aims to find a KG subgraph that can instantiate a given SPARQL based on \( K \). Firstly, we define an atomic subgraph \( q, type(q) \in \{\text{CVT, Single}\} \) in Freebase, where \( type(\cdot) \) is a function that identifies the type of an atomic subgraph. \( \text{CVT} \) is a Compound Value Type (CVT) subgraph and \( \text{Single} \) is a single-relation one. A CVT subgraph consist of a central CVT node and its corresponding one-hop edges. A single-relation subgraph can be formalized to a (subject, predicate, object) triple. Specifically, we use Virtuoso to store Freebase. Following Google’s instruction\(^6\), we classify the type of predicates in Freebase into single-relation and CVT. The difference is that the tail node of a CVT predicate is a CVT node and the entire CVT graph expresses an event, as the predicate “film.film_character.portrayed_in_films” shown in Figure 2 a. The single-relation one links two named entities, which means a fact.

In the training stage, given a predicate, we sample a set of corresponding atomic subgraphs in the database. For single-relation predicate, we simply construct “SELECT ?x WHERE ?x [predicate] ?y” to query KG and save all (subject name, predicate, object name) triples returned. For each CVT predicate, we sample a set of nodes that are the tails of the predicate, then instantiate a graph center at the node for each one in the set. As shown in Figure 2 a, the instantiated CVT graph consists of both inside and outside edges in one-hop connection. For example, in order to sample the graphs of predicate “film.film_character.portrayed_in_films”, we firstly construct a query, “SELECT ?x WHERE [?y film.film_character.portrayed_in_films ?x]”, to get corresponding CVT nodes, and then for each node (e.g. m:09r3hdx), we construct another query, “SELECT ?x WHERE [?y m:09r3hdx m:09r3hdx ?p_out.ORDER BY ?x LIMIT 1]”, to instantiate the CVT graph. Due to limited resources, instead of covering the entire predicates in Freebase, we selected a subset of predicates involved in the two datasets mentioned in Section 4.1.

\(^6\)https://github.com/google/Freebase-wikidata-converter
In the inference stage, given a SPARQL expression S, we search all variables (e.g., replacing “?x” with “?y ?x ?num” in the SPARQL) in KG and sample one from the results to instantiate a complete graph G. As shown in Figure 2 b, a SPARQL may consist of both single-relation and CVT subgraphs.

3.3.2 Subgraph Serialization. To explore how to mine the general statements more effectively for different entities with the same predicate, we propose two strategies of serialization.

Entity name. In this setup, we serialize the subgraph with the original entity name. As we know, Wikipedia is used to train PLMs, so that keeping the original name may be a simple but effective way to utilize the knowledge the model has learned. For single-relation subgraph, the serialization pattern is “<H> [head entity name] <D> [head entity description] <P> [predicate] <T> [tail entity name] <D> [tail entity description]”, the special tokens “<H>, <D>, <P>, and <T>” mean the head entity, description, predicate and tail entity, respectively. For CVT subgraph, we add a special token “R@” in front of inside predicates (point to a CVT node) in order to traverse the subgraph in a uniform format. Therefore, the serialization pattern of one edge in the subgraph is “<P> [R@][predicate] <T> [entity name] <D> [entity description]”, and then we just concatenate all edges together. Figure 2 gives an example.

Entity type placeholder. In this setup, we replace the entity name with its entity type in both input and target text. Intuitively, the delexicalization will make model focus more on predicate rephrasing. Previous work [8] picked the entity type that is mentioned most in the first sentence of the entity’s Wikipedia article. However, it does not make sense to fix the entity type in all contexts. Instead, we directly utilize the head and tail entity types contained in a Freebase predicate. The predicate in Freebase is organized as “domain:subject_type:property”, where we treat the “property” part as the type of object entity. For example, the part of serialization shown in the Figure 2 a will be: “<P> R@film:actor:film <T> Nick Cannon”.”

3.3.3 Collection of Subgraph Description. Given a atomic subgraph g, the auto-prompter is trained to generate the corresponding subgraph description, denote as d. Training the aforementioned auto-prompter needs large-scale labeled SPARQL and NL description pairs, since the SPARQL involves a lot of KB predicates, e.g., “film:film:actor”, and SPARQL operators, e.g., “ORDER BY ?num DESC LIMIT 1” (means “Argmax”). For example, given a non-NL SPARQL (a simplified version of the Figure 2 b) “SELECT DISTINCT ?x WHERE {m.01d1st ns:film:actor:film ?y. ?y ns:film:performance:film ?x . ?x ns:film:film:initial_release_date ?num .} ORDER BY DESC(?num) LIMIT 1” and the name of topic entity m.01d1st “Nick Cannon”, an annotator needs to understand the SPARQL and write the corresponding NL text “Nick Cannon star in film [?x]”. Labelling such large-scale data is impracticable. Therefore, the motivation here is to make use of large-scale unstructured corpus to fill the gap between predicates and natural language expression. Different from [8] who use a heuristic string matching rule to find the phrases in corpus by co-occurrence of entity names, we propose a novel soft-generation
approach by combining the distant supervision and the ability of modern generative PLMs together. Specifically, to find the NL descriptions for atomic subgraphs, we rely on the Wikipedia 2018-12-20 dump as the source of text documents. Each page in Wikipedia has a title and a content that consists of a list of paragraphs. All these fields are re-tokenized by Spacy\(^1\) and indexed by Elasticsearch\(^8\). As mentioned in [8, 22], we believe that the distant supervision assumption has been effective on Wikipedia. For single-relation subgraph \(g\), we match sentence \(d\) in Wikipedia if the subject name and the object name of this triple co-occur in the same sentence. For CVT subgraph \(g\), we find paragraphs \(d\) in Wikipedia if all entities of this subgraph co-occur in the same paragraph. In addition, we remove the sentences that do not have any entity in matched paragraphs and drop subgraphs that match nothing. Furthermore, during the training phase, the entity names in the descriptions are replaced by their types in the entity type placeholder setup, while in another setup, the descriptions remain untouched. Accordingly, we need to replace the entity types with the corresponding names in the inference phase to get the prompt text.

3.3.4 Training. The auto-prompter is based on an advanced generative PLM, BART\(^{12}\). Specifically, in order to make the optimization process more stable, we merge and shuffle the two kinds of subgraph as training data, making them evenly distributed in each mini-batch. Then we fine-tune two auto-prompters based on the two serialization strategies mentioned above. The details of the hyper-parameters setting are recorded in Appendix A.1. The auto-prompter is trained with a maximum likelihood objective. Given the training samples \((\bar{d}, g)\), the objective \(\mathcal{L}\) is defined as:

\[
\mathcal{L} = -\sum_{M} \sum_{i=1}^{M} \log p_{\theta}(\tilde{d}_i | \bar{d}_{<i}, g)
\]

(5)

where \(\tilde{d}_i\) is the \(i\)-th token in \(\bar{d}\) and the \(M\) is the length of training instances.

3.3.5 Prompt Text. The motivation for developing the prompt text is to smooth the transformation from non-NL SPARQL to NL question. Therefore we rephrase the formal expression in KB to a NL form with the help of PLMs. As shown in Figure 2 b, the auto-prompter successfully and correctly generates the relation, “A Very Scholl Gyrls Holla-Day starring Nick Cannon”, from a relative complex CVT subgraph. In a word, given a SPARQL expression \(S\), Section 3.3.1 instantiates and samples one RDF subgraph \(G\). Section 3.3.2 adds entity descriptions and serializes each \(g\) in \(G\). Lastly, the well-trained auto-prompter is used to generate prompt text \(T\) given \(G\). It is hard to evaluate the quality of prompt text by automatic metrics because there is no target text. Therefore we conduct human evaluations and report the scores in Section 4.7.

3.4 Question Generation

Question generation \(\mathcal{QG}(\cdot)\) takes both the SPARQL \(S\) and prompt text \(T\) as input, and generates the question \(\mathcal{Q}\) with max likelihood logit in the beam search space. The question generator is implemented based on another BART. The training and inference processes here are similar to the classic fine-tuning methods of BART. The overall process is shown in Figure 3.

3.4.1 Data Construction. We construct a training sample by concatenating \(S\) and \(T\) together as input and the annotated question as target. In order to better prompt the question generator, we add “(the [var])” behind the corresponding instantiated entity in prompt text for each variable in SPARQL\(^9\).

It is worth noting that, following previous works, we convert \(\mathcal{Q}\) to \(\mathcal{Q}_{\hat{Q}}\) by \(\Phi\) to get the prompt text.

3.4.2 Training and Inference. We fine-tune the question generator as a standard sequence-to-sequence model from the input to the output text, as presented in BART\(^{12}\). Formally, given the training sample \((S, T, \hat{Q})\), the objective \(\mathcal{L}'\) is defined as

\[
\mathcal{L}' = \sum_{N} \sum_{i=1}^{[\hat{Q}]} \log p_{\Phi}(\hat{Q}_{\hat{Q}} | S, T) - \sum_{N} \sum_{i=1}^{[\hat{Q}]} \log p_{\Phi}(\hat{Q}_{\hat{Q}} | S, T)
\]

(6)

where \(\hat{Q}_{\hat{Q}}\) is the \(i\)-th token in the target question \(\hat{Q}\). \(N\) is the length of training samples and \(\Phi\) denotes the parameters of the question generator.

We use the standard beam search method for decoding. The specific parameter settings are reported in Appendix A.1.

4 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of our proposed approach.

4.1 Dataset & Preprocessing

WQCM/WQ1.1, WebQuestionsSP [30] and ComplexWebQuestions 1.1 [27] are widely used question answering datasets that contain natural language questions and corresponding SPARQL queries. Previous works [5, 9, 11] combined WebQuestionsSP and ComplexWebQuestions (the older version) and convert the SPARQL from SELECT query to CONSTRUCT query to get RDF graphs.

\(^1\)https://spacy.io

\(^8\)https://www.elastic.co/elasticsearch

\(^9\)For example, add “(the ?x)” behind “A Very School Gyrls Holla-Day”

---

**Figure 3: Overview of question generation.**

**Table 1: Statistics of the datasets.**

| Dataset       | #Rel | #Instances Total | Length |
|---------------|------|------------------|--------|
| WQCM/WQ1.1    | 931  | 29,909 / 2,529 / 2,529 | 34,967 | 14.12 |
| w/Ops         | 494  | 4,097 / 300 / 343   | 4,740  | 14.69 |
| w/o Ops       | 870  | 25,812 / 2,229 / 2,186 | 30,277 | 14.03 |
| PathQuestions | 378  | 9,793 / 1,000 / 1,000 | 11,793 | 12.4  |

---

\[\text{Q: what film did nick cannon star in that was released latest?}\]
We choose the following two categories of models as our baselines: whether containing complex operations (OPs), and the average works only released the processed results without original SPARQL expressions, wherein the exact semantics are lost. Therefore, we merge and reprocess these two datasets by ourselves and release a new dataset, WQCWQ1.1. We leave the original training set untouched and randomly divide the validation/test set equally.

PathQuestions (PQ) is constructed from a question answering dataset [31]. The main characteristic is that PQ only consists of chain questions, wherein the path between the topic entity and the answer is 2-hop or 3-hop. Compared with WQCWQ1.1, PQ does not contain any complex operation in questions. We used the same data as the existing works [5, 11].

Brief statistics of the datasets are listed in Table 1, including the total number of relations, the data split, the subset divided by whether containing complex operations (OPs), and the average length of questions.

4.2 Implementation

Our auto-prompter and question generator are both based on pre-trained models BART [12]. We initialize our model weights with the BART-base checkpoint released by HuggingFace’s Transformers [29]. We follow BART to use BytePair Encoding (BPE) vocabulary [19] with the size of 50,265 and adopt Adam [10] as the optimizer. Since the computational resources are limited, we pick a set of hyper-parameters and train the auto-prompter on unsupervised data for 10 epochs. It took 60 hours on 4 NVIDIA A100 (40GB) GPUs. For training question generator, we apply different hyper-parameters on each data proportion setting. More details, including the hyper-parameters and search space settings, are reported in Appendix A.1.

4.3 Baseline Methods

We choose the following two categories of models as our baselines:

**Pre-trained Models.** We chose JointGT as the pre-trained baseline. JointGT [9] is a BART-based model for KG-to-text generation. It adopts a structure-aware semantic aggregation module to model the structure of an input graph at each Transformer layer. Afterward, it is pre-trained in large-scale KG-to-text corpora, and then fine-tuned in the downstream tasks, including question generation.

**Task-Specific Models without Pre-training.** We also adopted the recent task-specific computational models without pre-training as baselines, including Graph2Seq [5] and KTG [3]. Graph2Seq introduces a bidirectional graph encoder to model subgraphs in KG and generate questions by a graph-to-sequence generator with a coping mechanism. KTG proposes a knowledge-enriched, type-constrained, and grammar-guided model with auxiliary information to enrich input.

We report the baseline results directly if they use the same dataset as ours. Otherwise, we implement these baselines using the codes and parameters released by the original papers.

4.4 Automatic Evaluation Metrics

Following previous QG works [3, 5, 9, 11], we use BLEU-4 (B-4) [18], METEOR (ME) [7] and ROUGE-L (R-L) [13] as our evaluation metrics. Initially, BLEU-4 and METEOR were designed to evaluate machine translation systems, and ROUGE-L was designed to assess text summarization systems.

4.5 Experimental Results

4.5.1 Automatic Evaluation. Table 2 shows the detailed evaluation results comparing our proposed models against other state-of-the-art baselines in six data proportion settings, from 0.1% to 100%, respectively. AutoQGS is implemented on the entity name serialization setup by default, and we also report the experimental results on entity type placeholder serialization setup on WQCWQ1.1, denoted as AutoQGS-T. Since AutoQGS performs better than AutoQGS-T under most settings on WQCWQ1.1, we only report AutoQGS performance hereafter. As we can see, both AutoQGS and AutoQGS-T outperform every baseline by a large margin on WQCWQ1.1, particularly in few-shot settings. Table 3 shows the mean average gains across six settings. AutoQGS generally exceeds Graph2Seq/JointGT by 12.88/8.06 BLEU-4 points in WQCWQ1.1, and 29.82/3.43 in PQ, respectively. Specifically, for Graph2Seq, the vocab depends on the training data heavily. As the training instances decrease, the vocabulary becomes smaller and more words become OOV (out of vocabulary), which severely degrades the performance of this model. Furthermore, our model outperforms JointGT in all six settings except in 0.1% of PQ. We speculate that, in the 0.1% setting, there are only ten training instances (9793*0.001, we take the upper bound), which is too difficult for both JointGT and AutoQGS. These results verify that AutoQGS can effectively and accurately generate questions based on SPARQL in low-resource scenarios.

4.5.2 Impact of Complex Operations. Next, we evaluate how the complex operations affect the performance of our AutoQGS. We keep the same divisions in WQCWQ1.1 and split data into two subsets by whether containing complex operations. The statistics are reported in Table 1. About 86.59% of the data do not contain any complex operation, which means that this part is similar to the data used by previous works. We report the comparison results on both subsets in Table 4. Results show that AutoQGS performs better in both settings. The results confirm the point that our model can generate questions from SPARQL better than others.

4.6 Ablation Test

We conduct ablation tests to investigate the effectiveness of AutoQGS in four representative data proportion settings (0.1%, 1%, 10%, 100%) by removing the prompt text and replacing the prompt text with topic entity descriptions (re/desc, for short) one at a time. As shown in Table 5, removing the prompt text leads to significant performance reduction in all settings, particularly in 0.1% and 1%. The decline is consistent with our purpose as the prompt text is designed to enable the model to perform better in low-resource scenarios. Similarly, we also observe performance decline by replacing prompt text with topic entity descriptions. This result again validates the effectiveness of prompt text, as it is also developed to mine relations between entities rather than only describe them.

4.7 Human Evaluation

To further evaluate AutoQGS, we conduct two human evaluations on the auto-prompter and question generator results, respectively.

4.7.1 Human Evaluation of Auto-Prompter. We conduct two human evaluations, as shown in Table 6. The main purpose of the Auto-Prompter is to help the question generator generate more readable questions. Therefore, we first remove the prompt text and then replace it with topic entity descriptions (re/desc, for short) one at a time to evaluate the effectiveness of the prompt text.

4.7.2 Human Evaluation of Question Generator. We also conduct two human evaluations, as shown in Table 7. The main purpose of the question generator is to generate readable questions. Therefore, we first remove the prompt text and then replace it with topic entity descriptions (re/desc, for short) one at a time to evaluate the effectiveness of the prompt text.
Table 2: Results on WQCWQ1.1 and PathQuestions in six data proportion settings. The results marked with †, ‡ and † are re-printed from the references [3], [5] and [9], respectively.

| Model          | WQCWQ1.1 | PathQuestions |
|----------------|----------|---------------|
|                | 0.1%     | 0.5%          | 1%            | 5%         | 10%       | 100%       |
|                | B-4      | ME            | R-L           | B-4        | ME        | R-L        | B-4        | ME        | R-L        | B-4        | ME        | R-L        | B-4        | ME        | R-L        |
| Graph2Seq      | 0.00     | 0.87          | 5.98         | 7.01       | 12.65     | 26.86      | 8.53       | 13.51     | 31.70      | 17.17      | 21.56     | 45.11      | 19.69      | 23.25     | 46.04      | 29.56      | 31.14     | 58.34      |
| JointGT        | 6.92     | 18.77         | 32.14        | 11.31      | 25.10     | 36.62      | 13.83      | 24.90     | 42.52      | 20.79      | 28.41     | 49.67      | 25.25      | 30.50     | 54.05      | 32.81      | 34.48     | 60.92      |
| AutoQGS        | 15.26    | 22.07         | 42.97        | 21.56      | 27.06     | 48.70      | 24.09      | 28.77     | 51.04      | 29.58      | 32.43     | 56.90      | 31.81      | 33.82     | 59.07      | 36.93      | 36.63     | 63.82      |
| AutoQGS-T      | 14.43    | 21.45         | 42.59        | 20.40      | 26.09     | 47.53      | 23.18      | 28.11     | 50.40      | 28.86      | 31.83     | 56.13      | 32.00      | 33.79     | 59.52      | 36.49      | 36.38     | 63.53      |

Table 3: Average gains on WQCWQ1.1 and PathQuestions over six data proportion settings.

| Dataset | WQCWQ1.1 | PathQuestions |
|---------|----------|---------------|
|         | B-4      | ME            | R-L           | B-4        | ME        | R-L        | B-4        | ME        | R-L        | B-4        | ME        | R-L        | B-4        | ME        | R-L        |
| Graph2Seq | 12.88    | 12.97         | 17.98        | 29.82      | 18.83     | 26.22      | 8.06       | 3.44      | 7.43       | 3.43       | 0.15      | 0.63       |
| JointGT   | 8.06     | 3.44          | 7.43         | 8.06       | 3.44      | 7.43       | 3.43       | 0.15      | 0.63       |

Table 4: Results on WQCWQ1.1 subsets dividing by whether containing complex operations.

| WQCWQ1.1       | w/ OPs               | w/o OPs             |
|----------------|----------------------|---------------------|
| Model          | B-4      | ME            | R-L               | B-4        | ME        | R-L               |
| Graph2Seq      | 12.21    | 26.63         | 51.72             | 30.11      | 31.32     | 58.79             |
| JointGT        | 24.33    | 30.83         | 54.35             | 35.08      | 35.35     | 62.48             |
| AutoQGS        | 35.75    | 36.73         | 62.11             | 36.03      | 36.09     | 63.37             |

Table 5: Ablation tests on WQCWQ1.1

| Data Proportion | 0.1% | 1% |
|-----------------|------|----|
| Model           | B-4  | ME | R-L | B-4 | ME | R-L |
| AutoQGS w/o prompt text re/desc | 15.26 | 22.07 | 42.97 | 24.09 | 28.77 | 51.04 |
| AutoQGS w/o prompt text re/desc | 31.81 | 33.82 | 59.07 | 36.93 | 36.63 | 63.82 |

Considering that the goals of the two components are essentially similar, we decide to run the same two criteria following [8].

**Predicates identification.** Annotators were asked to estimate whether the generated text expresses all predicates in the given SPARQL (or subgraph) or not.

**Naturalness.** Annotators were requested to assign each generated text based on the fluency and readability by a score from 1 to 5, where (5) perfectly clear and natural, (3) grammatically correct but seems artificial, and (1) entirely not understandable.

For the evaluation of auto-prompter, we randomly sample 100 atomic subgraphs from the dataset. For the evaluation of question generator, we randomly sample 100 instances from the test set. We also collect the outputs from the most competitive baseline, JointGT, for comparison. All evaluations are done with the help of 3 annotators. Results in Table 6 show some critical observations. First of all, the auto-prompter has the ability to paraphrase predicates consistently and fluently. On the other hand, the question generator achieves remarkable results in all four settings, whereas the naturalness score in the 100% setting is even higher than that of annotated questions (denoted as Golden). Generally speaking, our AutoQGS can beat the corresponding baselines in both predicates identification and naturalness.

### 4.8 Case Study

To provide a complete and visual presentation of AutoQGS, we provide a case in Figure 4. Steps 1 to 4 display the processes of how to get the prompt text by the auto-prompter. Firstly, we re-write the SPARQL query (adding "?c ?num" in this case) to search all variables in KG and sample one from the results to instantiate a
1. SPARQL

```sparql
SELECT DISTINCT ?x WHERE {
  ?x ns:m.06y57 ns:travel.travel_destination.tourist_attractions .
  ?x ns:exhibitions.exhibition_venue.exhibitions_at_this_venue .
  ?c ns:exhibitions.exhibition_run.opened_on ?num .
} ORDER BY ?num LIMIT 1
```

2. Sampled instantiated subgraph

```
{Sydney} travel.travel_destination.tourist_attractions
{Powerhouse Museum} ns:exhibitions.exhibition_venue.exhibitions_at_this_venue
{1999-12-07} ns:exhibitions.exhibition_run.opened_on
```

3. Serialized subgraphs

```
<Hi> Sydney <Do> Sydney is the state capital of New South Wales and the most populous city in Australia and Oceania.
<Pi> travel.travel_destination.tourist_attractions
<Ti> Powerhouse Museum <Do> The Powerhouse Museum is the major branch of the Museum of Applied Arts & Sciences in Sydney.  
<Pi> exhibitions.exhibition_run.opened_on <Ti> 1999-12-07 <Do> No description.
```

Q: what is the exhibition venue that opened the earliest in Sydney, Australia?

| Data Proportion | Model     | Generated Question                                                                 |
|-----------------|-----------|-------------------------------------------------------------------------------------|
| 0.1%            | JointGT   | what structure constructed in 1970 is there to see in the city’s powerhouse museum? |
|                 | AutoQGS   | what attractions are there to see in Sydney?                                       |
| 1%              | JointGT   | what exhibition venue that opened on 1999 is in Sydney, England?                    |
|                 | AutoQGS   | what exhibition venue that opened prior to 1999 is in Sydney, Australia?            |
| 10%             | JointGT   | what is there to do in Sydney that opened on December 7, 1999?                      |
|                 | AutoQGS   | what exhibition venue in Sydney opened the earliest?                                |
| 100%            | JointGT   | what exhibition venue in Sydney opened on December 7, 1999?                         |
|                 | AutoQGS   | what is the earliest opened exhibition venue in Sydney, Australia?                  |

4. Prompt text

```
The Powerhouse Museum opened on 7 December 1999.
```

**Figure 4: Case study**

4.9 Error Analysis

Table 7 shows some failure cases on the WQCWQ1.1 test set in 100% data proportion setting. For convenience, variables in SPARQL are instantiated and put together with their variable names. The most common mistake for the auto-prompter is that the distant supervision approach sometimes produces descriptions that are not consistent with the facts, as shown in the first case. On the other hand, one of the frequent error patterns we find out for the question generator is incorrect statement. For instance, in the second example, the statement "what type of government is used" is grammatically correct, but the usage is inappropriate. Another error pattern is lack of knowledge. In the first example, given the SPARQL and the prompt text, we still don’t know that Caesar was stabbed. But it is usual to include common sense in a question.

---

10 In this example, the Theatre of Pompey was built during the latter part of the Roman Republican era by Pompey the Great, not Julius Caesar.
Table 7: Error analysis

| SPARQL | Prompt text | Golden | AutoQGS |
|--------|-------------|--------|---------|
| SELECT DISTINCT ?x WHERE { "Julius Caesar" people.deceased_person.place_of_death "The Theatre of Pompey"(the ?x) . } | The Theatre of Pompey (the ?x) was built during the reign of Julius Caesar. | where was caesar when he was stabbed? | where did julius caesar die? |
| SELECT DISTINCT ?x WHERE { "Chile" location.country.form_of_government "Presidential system"(the ?x) . "Presidential system"(the ?x) government.form_of_government.countries "Brazil" . } | Chile has a Presidential system (the ?x) with a bicameral legislature. Brazil is a country with a Presidential system (the ?x) | what are the government types of chile and brazil? | what type of government is used in both brazil and chile? |

4.10 Data Augmentation

 Automatically constructing question-answer pairs from knowledge bases is one of the objectives of question generation. Based on the WQCWQ1.1 dataset, we augment data by replacing topic entities in SPARQL. Specifically, given a SPARQL, we construct a new one to get a list of different topic entities by replacing the topic entities with variables. For each set of entities in the result, we construct another SPARQL, in which only the named entity is different from the original one, to form a new sample. Afterward, we use the best AutoQGS trained in 100% data to generate the corresponding questions. In order to multiply the data tenfold, we randomly choose ten instances from querying results for each item in WQCWQ1.1. Finally, we generate a corpus of 340K question-SPARQL pairs for future research on question generation and question answering.

5 CONCLUSION

 In this paper, we propose AutoQGS, an auto-prompt approach for low-resource KBQG from SPARQL. Firstly, we generate questions directly from SPARQL to handle complex operations. Secondly, we propose an auto-prompter to rephrase SPARQL into NL prompt text, smoothing the transformation from non-NL SPARQL to NL question with PLMs. Lastly, we devise a question generator to generate questions given SPARQL and corresponding prompt text. Experimental results show that our approach achieves state-of-the-art performance especially in low-resource scenarios. Furthermore, we generate a dataset of 330k factoid complex question-SPARQL pairs for further KBQG research.

ACKNOWLEDGMENTS

 This work was supported by the National Key Research and Development Program of China under Grant No. 2020AAA0108600 and No. 2020YFC0833301.

A APPENDIX

A.1 Hyper-Parameter Setting

Table 8: Hyper-parameter search space. [ ] indicates that the listed numbers will be traverse one by one.

| Model | auto-prompter | question generator |
|-------|---------------|--------------------|
| Hyper-parameter | Setting | Search Space |
| Learning Rate | 3e-5 | [3e-5, 5e-5, 1e-4] |
| Warmup Proportion | 0.1 | [0.1, 0.2, 0.3] |
| Batch Size | 64 | [16, 24, 32, 64] |
| Beam Size | 10 | [5, 10] |
| Length Penalty | - | [1.0, 1.2] |
| Input Length | 512 | 512 |
| Output Length | 512 | 128 |
| Warmup Epoch | 10 | 50 |
| Early Stop Patience | - | 10 |
| Maximum Gradient Norm | 1.0 | 1.0 |
| Optimizer | Adam | Adam |
| Epsilon (for Adam) | 1e-8 | 1e-8 |

Table 9: Best assignments of hyper-parameters for question generator.

| Dataset | WQCWQ 1.1 |
|---------|-----------|
| Data Proportion | 0.1% 0.5% 1% 5% 10% 100% |
| Warmup Proportion | 0.2 0.1 0.2 0.2 0.2 |
| Batch Size | 16 16 32 32 32 64 |
| Beam Size | 10 10 10 10 10 |
| Length Penalty | 1.2 1.2 1.2 1.0 1.0 1.0 |

We provided the detailed settings of hyper-parameters for training the auto-prompter and the question generator in Table 8. The table include the hyper-parameter settings for training auto-prompter and the hyper-parameter search space for training question generator. We implement our models base on Huggingface’s Transformers [29]. For auto-prompter, we train the model on unsupervised data for 10 epochs. For question generator, We set Warmup Epoch, which means the expected training rounds, to control warmup steps. The hyper-parameter search space and the best assignments are listed in Table 8. We go through each combination of hyper-parameters in every few-shot settings to find the optimal one. The selection criterion is BLEU-4 on the validation set.
