# A Deep Neural Approach to KGQA via SPARQL Silhouette Generation

Sukannya Purkayastha*†
TCS Research, India
sukannya.purkayastha@tcs.com

Saswati Dana†, Dinesh Garg, Dinesh Khandelwal, G.P Shrivatsa Bhargav
IBM Research, India
{sdana027, garg.dinesh, dikhand1, gpshri27}@in.ibm.com

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9892263

***Abstract*—Knowledge Graph Question Answering (KGQA) has become a prominent area in natural language processing due to the emergence of large scale Knowledge Graphs (KGs). Semantic parsing based approach is the predominant direction to solve the KGQA task where natural language question is translated into a logic form such as SPARQL query. Recently Neural Machine Translation (NMT) based approaches are gaining momentum in order to translate natural language query to structured query languages thereby solving the KGQA task. However, most of these methods struggle with out-of-vocabulary words where test entities and relations are not seen during training time. In this work, we propose a modular two stage neural architecture to solve the KGQA task. Stage-I of our approach comprises a NMT-based seq2seq module that translates a question into a sketch of the desired SPARQL query called a SPARQL silhouette. Stage-II of our approach comprises a Neural Graph Search (NGS) module which aims to improve the quality of the SPARQL silhouette by detecting the right relations in the underlying knowledge graph. Experimental results show that we achieve substantial improvements and obtain state-of-the-art performance or comparable results to the best performing systems on two benchmark datasets. We believe, our proposed approach is novel and will lead to dynamic KGQA solutions that are well-suited for practical applications.**

***Index Terms*—KGQA, entity/relation linking, SPARQL Silhouette, seq2seq model, Neural Graph Search Module,**

## I. Introduction

A Knowledge Graph (KG) is a large collection of real world facts that are stored in the form of triples such as $\langle Joe\ Biden, president, United\ States \rangle$, where, "*Joe Biden*" and "*United States*" are the entities and "*president*" is the relation between them. The task of Knowledge Graph Question Answering (KGQA) is an important application where a system is required to answer a natural language question by leveraging the facts present in the given KG. A KGQA system permits users to retrieve information from a KG without any prior knowledge about the KG schema or query languages such as SPARQL, SQL, etc. The availability of large-scale KGs, such as Freebase [1], DBpedia [2], YAGO [3], NELL [4], Google's Knowledge Graph [5], and their applicability in various business applications have made KGQA an important research area within NLP.

There are several approaches proposed to solve the KGQA task. These approaches can be grouped into two broad categories:

1) *Semantic parsing-based*: In these approaches [6]–[8], a natural language question is first transformed into a structured query language or logic form such as SPARQL, SQL, $\lambda$-DCS [9], CCG [10]. The generated query is then executed against the given KG to get the answer of the given question.
2) *Information extraction-based*: These approaches [11]–[13] extract a subgraph from the underlying KG which depends on the entities/relations present in the question. Next, they perform graph-based reasoning on the subgraph to reach the final answer directly without generating any intermediate logic form.

The popular semantic parsing-based approaches for KGQA [14]–[16] are inherently modular and pipelined in nature because they decompose the problem of logic form generation into several subtasks: (i) question understanding, (ii) linking mentions in the question text to the entities and relations in the KG, (iii) generating the final logic form with various constraints required to get the answer. Some of these modules, for example, entity/relation linking, are generally far from being perfect and hence induce a noticeable amount of noise in the pipeline. Because of this, such approaches [17] end up handling simple questions well where a single KG fact is needed to answer the question. These approaches, however, struggle while handling complex questions [18]–[20] that require multiple facts. For complex questions, one of the key challenges lies in the large sized search space when linking entities and relations. Other drawback of the pipeline-based approaches involve propagation of the noise from upstream modules into downstream modules where, usually there is no explicit provision to correct the noise.

NMT based approaches [21] have been emerging as an alternative approach [22], [23] for semantic parsing with a hope of alleviating the limitations of pipelined based approaches. These approaches are good at syntactic and semantic understanding of the complex questions. However, NMT-based approaches have their own limitations - (i) they require large amount of training data, (ii) they cannot handle unseen entities/relations at test-time due to their fixed vocabulary. Motivated by these limitations, we propose a novel *two-stage neural* approach for KGQA (see Figure 1). This approach embraces the best of both the worlds – (i) using NMT for handling complex questions, (ii) using masking technique

* Work done while the author was an intern at IBM Research
† Equal contribution

978-1-7281-8671-9/22/$31.00 ©2022 IEEE

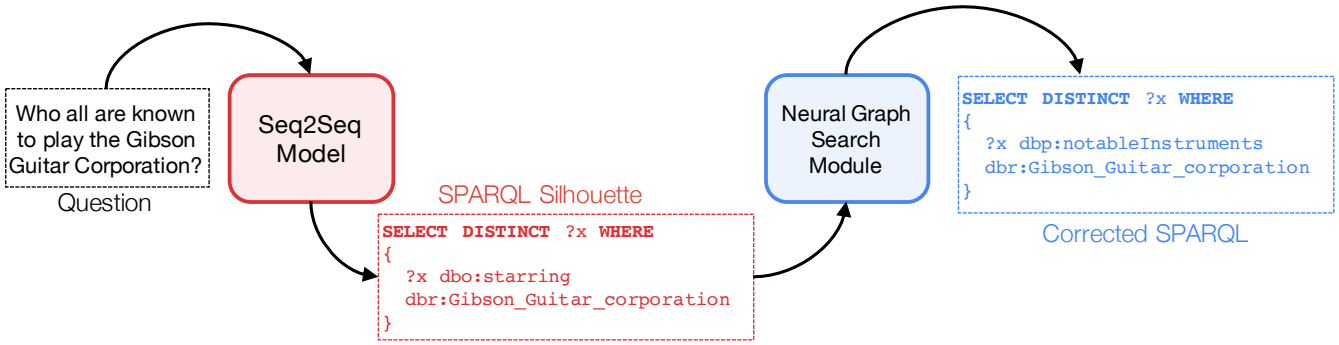


Fig. 1: Overview of our proposed two stage approach.

with entity/relation linking module to handle unseen entities/relations.

The Stage-I of our approach generates a sketch of the target SPARQL for a given natural language question, we call this sketch a *SPARQL silhouette*. In this stage, we propose three different masking schemes to mask all the entities and relations present in the input question. We however handle the entity/relation linking separately via an off-the-shelf entity/relation linker. We further leverage the masking scheme to simulate the noise level in entity/relation linking process for the purpose of ablation studies. Stage-II comprises a *Neural Graph Search (NGS)* module. This module takes the SPARQL silhouette as an input and reduces noise introduced by the entity/relation linker in Stage-I. To demonstrate the effectiveness of our approach, we simulate three levels of noise scenarios in linking; *noise-free*, *partly-noisy* and *fully noisy*. We show that, there is a substantial gap between performance numbers of ideal and realistic linking scenarios. Finally integrating Stage-II module with Stage-I boosts the performance significantly in the realistic scenario to achieve state-of-the-art or competitive performance as compared with the previous systems on two benchmark datasets.

## II. Related Work

The vast body of literature that exists on the KGQA task is nearly impossible to cover. One can refer to the survey papers [24] for an in-depth account of KGQA literature.To a large extent, semantic parsing is the predominant paradigm in the KGQA literature. Semantic parsing-based approaches can be classified into two categories: (i) *neural*, and (ii) *non-neural*. Non-neural approaches [25]–[27] are somewhat dated now and use handcrafted features and rules. As far as neural semantic parsing approaches are concerned, they can be further divided into two subcategories: (i) *Non-NMT-based*, (ii) *NMT-based*.

### A. Non-NMT-based Approaches

Pipeline-based approaches [14]–[16] break the problem of semantic parsing a complex question [20], [28]–[30] into more manageable subtasks such as question understanding, entity/relation linking, logic form generation and use reusable modules for solving the subtasks. In these approaches, all intermediate modules need not be neural-based. Each of these submodules introduces its own errors, which propagate to the downstream pipeline. Another line of work [31]–[35] maps the problem of semantic parsing on KG to a query graph (a subgraph of KG) generation, which can be easily translated into the SPARQL.

### B. NMT-based Approaches

In the last few years, NMT-based approaches are being used to translate natural language questions into SQL [36]–[38]. Recently, [22] compared various types neural architecture and showed that CNN-based seq2seq model performs best to generate SPARQL queries from natural language questions. One limitation of their approach is that output vocabulary for SPARQL generation is limited to the entities/relations seen during training. As a result, their performance reduces drastically if the overlap of entities and relations in the training and test sets differ. This motivated us to incorporate masking approach to efficiently handle unseen entities/relations during test time. To the best of our knowledge, our work is the first of its kind in solving KGQA task which considers multiple relations and uses NMT based approach that can handle unseen entities/relations by proposed noise simulator with various masking strategies.

## III. The KGQA Task

In KGQA, we are given a Knowledge Graph $\mathcal{G}$ comprising of an entity set $\mathcal{E}$, a relation set $\mathcal{R}$, and a set of knowledge facts $\mathcal{F}$. The knowledge facts are expressed in the form of triples; $\mathcal{F} = \{\langle e_s, r, e_o \rangle\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $e_s \in \mathcal{E}$ is known as *subject* or *head* entity, $e_o \in \mathcal{E}$ is known as *object* or *tail* entity, and $r$ is a relation which connects these two entities. These entities (relations) form the nodes (edges) of the KG. The task now is to identify the subset of entities from $\mathcal{E}$ that constitute the answer of a given question $Q$ in the natural language form. The most common family of approaches for the KGQA task is *semantic parsing* where, the given question $Q$ is first translated into a SPARQL which is then executed over the KG so as to get the answer set. For developing a system to convert a question into the corresponding SPARQL query, we are given a set of training data $\{Q_i, S_i, A_i\}_{i=1}^{n}$, where $Q_i$ is a question

(in natural language text), $S_i$ is the SPARQL query, and $A_i$ is the answer set obtained by executing $S_i$ on $\mathcal{G}$.

In this paper, we propose a two-stage system for KGQA. In Stage-I, seq2seq module generates a SPARQL silhouette with specific entities. Relations predicted in this module are corrected by the *neural graph search module* in Stage-II.

## IV. Stage-I: Seq2Seq Model

Figure 2 shows various components of Stage-I of our approach. We use external entity/relation linker to map surface form mentions and linking. Based on the suggestion of linker, *noise simulator* masks mentions and entities/relations in the input question text and corresponding gold SPARQL for the training data. *Noise simulator* employs different masking schemes depending on the desired level of noise that we wish to simulate. The masked question is used as an input to the CNN-based sequence-to-sequence (seq2seq) module which converts it into a SPARQL silhouette. Note, seq2seq models have achieved state-of-the-art performance in machine translation task [39] and they can be based on RNN, CNN, or Transformer architectures. Our preliminary experiments showed that CNN-based model performs (F1 score $21.46\%$ for transformer compared to $25.59\%$ for CNN)[1] better than the transformer based model. This behavior is consistent with the prior research [22] that has shown that CNN-based seq2seq model performs best for translating natural language to SPARQL query. Hence, we opted for a CNN-based seq2seq model as our base model for Stage-I.

### A. Noise Simulator and Masking

The purpose of designing noise simulator module is two-fold: (i) To mask mentions and entities/relations in the question text as well as SPARQL, (ii) To simulate varying levels of noise in the entity/relation linking process. Masking helps us in two ways: (i) handling test entities/relations that are unseen during training, (ii) reducing vocabulary size as KGs contain a large number of entities and relations. A simple neural seq2seq model which translates natural language question into a SPARQL query will struggle to output some of the entities/relations during test time that are unseen during training time and hence will not be available in the output vocabulary. In the absence of linking and masking, our elementary experiment finds the performance of seq2seq model to be quite low ($F_1$ score $16\%$). Table I further corroborates this behavior where we have captured the statistics about % of entities and relations (i.e. properties and ontology in DBpedia) in validation and test sets that are seen in the training set. This suggests that entity/relation linker along with masking is an important step for the seq2seq model. Here, we propose three different types of masking schemes (Scenario A, B and C) and describe them in subsequent subsections .

**Scenario 'A', Noise-Free Linking:** In this scenario, we simulate an entity/relation linker that has $100\%$ $F_1$. We extract all the entities/relations from the gold SPARQL and assume these as output of the linker (see Figure 3). Next we align these entities (*dbr*) and relations (*dbp* and *dbo*) with the *surface-form* in the given question. We observe that entities match exactly with substrings in the questions most of the time (e.g. *Austin College* in Figure 3). For relations, other than substring match, we considered semantic similarity; e.g., a given relation `dbo:film` is semantically best aligned to word *movies* in the question. We use pre-trained fastext embeddings [40] to represent words and relations and compute cosine similarity between each word in the question and the given relation. The highest-scoring word is considered as the aligned word.

**Scenario 'B', Partly Noisy Linking:** Purpose of scenario 'B' is to allow partial noise in the entity/relation linking process. For this, we first feed the natural language question into an external entity/relation linker. The linker returns two objects: (i) A set of surface form mentions for entities/relations in the question text, and (ii) Linked entities/relations for these mentions. We take linker's output and find intersection of these entities/relations with the entities/relations present in the gold SPARQL. These common entities/relations are masked in the SPARQL. Also, their corresponding surface forms are masked in the question text. In order to mask the surface form in the question, we use exact match and string overlap based *Jaccard similarity*. Figure 4 illustrates this scenario.

**Scenario 'C', Fully Noisy Linking:** The goal here is to simulate a completely realistic scenario where we rely entirely on an external entity/relation linker. For this, we feed input question to the entity/relation linker and get the suggested surface form mentions and linked entities/relations. We mask these suggested mentions using exact match and partial match. Corresponding SPARQL query's entities/relations are also masked based on the suggestions. This scenario is depicted in Figure 5.

TABLE I: % of the entities and relations in val and test sets that are available within train set's gold SPARQLs.

| Dataset | Statistics | Val | Test |
|---|---|---|---|
| LC-QuAD-1 | Entities (dbr) | 52.3 | 46.8 |
| | Properties (dbp) | 97.2 | 98.3 |
| | Ontologies (dbo) | 96.5 | 94.6 |
| QALD-9 | Entities (dbr) | 27.1 | 25.9 |
| | Properties (dbp) | 0.0 | 16.9 |
| | Ontologies (dbo) | 47.8 | 38.3 |

### B. Convolutional `Seq2Seq` Model

The pair of masked question and SPARQL query obtained from the noise simulator is fed to a *Convolutional Neural Network (CNN)* based *seq2seq* model [41]. As shown in Figure 6, this model reads the entire masked question and then predicts the corresponding masked SPARQL query token-by-token in a left-to-right manner. This seq2seq model consists of the following key components.

**Input Embedding Layer:** Both encoder and decoder consist of an embedding layer that maps each input token to a

[1]Because of space constraint, supporting results have been put within parentheses

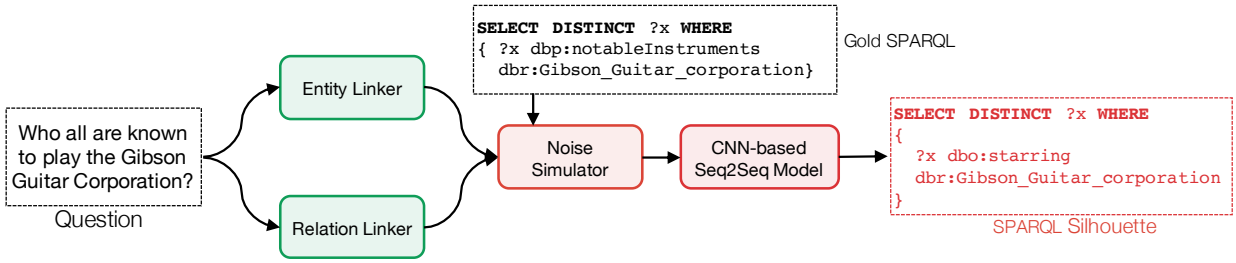


Fig. 2: Components of Stage-I.

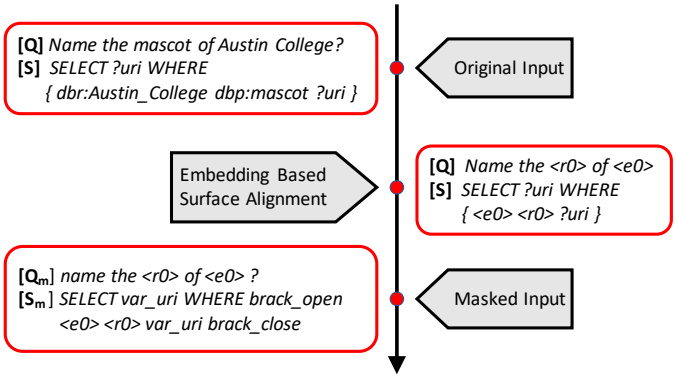


Fig. 3: An illustrative example for Scenario 'A': Noise-Free Linking

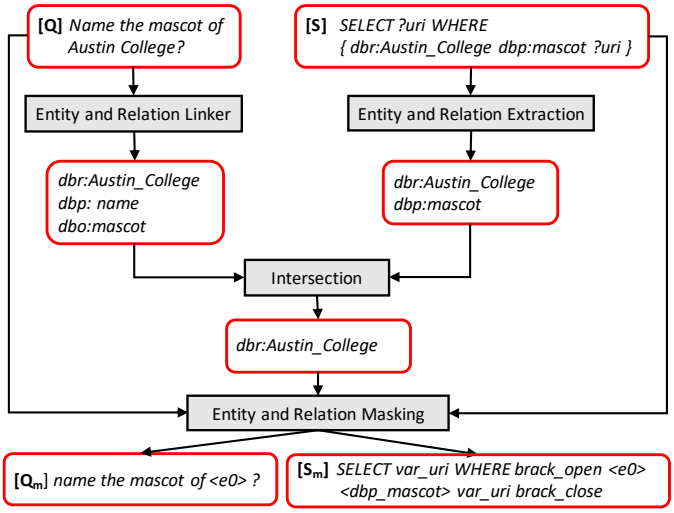


Fig. 4: An illustrative example for Scenario 'B': Partly Noisy Linking

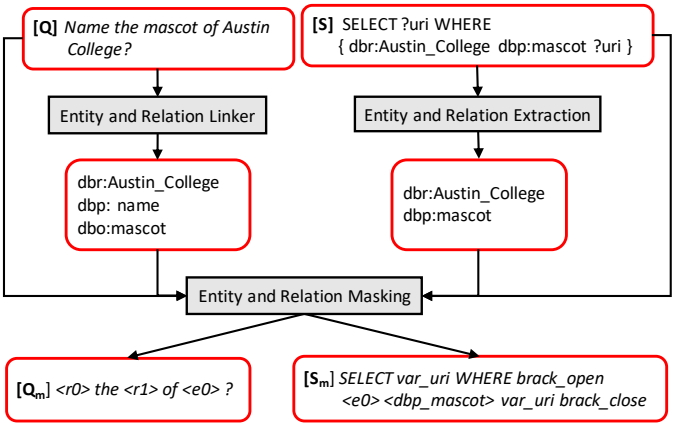


Fig. 5: An illustrative example for Scenario 'C': Fully Noisy Linking

point-wise summation of its word embedding and positional embedding. The embedding of each word is initialized randomly. In order to capture the sense of order, the model is provisioned with the positional embedding.

**Convolution + Pooling Layers:** The token embeddings obtained from the previous layer are fed to the multiple convolution and pooling layers. Each convolution layer consists of a 1-dimensional convolution followed by Gated Linear Units (GLU) [42]. Residual connections [43] are added from input to the output of each convolution layer.

**Multi-Step Attention:** Each decoder layer comprises a convolution layer followed by a multi-step attention layer. This multi-step attention is used to find the attention scores from a particular decoder state to the source tokens. Attention between decoder state $d_i$ (after $i^{th}$ layer) of the last token in generated sequence so far and state $z_j$ of the $j^{th}$ source element (after last encoder layer) is computed as: $a_j^i = exp(d_i \cdot z_j)/\sum_{t=1}^{m} exp(d_i \cdot z_t)$ where, $m$ is the number of source elements. The context vector, $c_i$, is now computed as, $c_i = [\sum_{j=1}^{m} a_j^i(z_j + e_j)] + d_i$ where, $e_j$ is the input embedding for the source element $j$.

**Output Layer:** Finally, output at a particular time step is calculated over all the $Z$ possible tokens, $P(z_{t+1}|z_1, \ldots, z_t, X) = softmax(Wd_L + b)$ where $P(z_{t+1}|\cdot) \in \mathbb{R}^Z$, and $W$, $b$ are trainable parameters. $d_L$ is the decoder state of last target element at the last layer $L$. $X$ is the input sequence.

**Training Loss:** The model is trained using *label smoothed cross-entropy loss* given by following expression (for single training example) $L(\theta) = -(1/N) \cdot \sum_{n=1}^{N} \sum_{z=1}^{Z} q(y_n = z|y_{n-1}) \cdot \log P_\theta(y_n = z|y_{n-1})$ where, $N$ is the number of words in output sequence and $y_n$ is the first $n$ tokens of output sequence. $P_\theta(y_n = z|y_{n-1})$ is model's probability to output token $z$ given $y_{n-1}$ sequence generated so far. The quantity $q(y_n = z|y_{n-1})$ is equal to $\gamma$ if $f(y_n) = z$ and

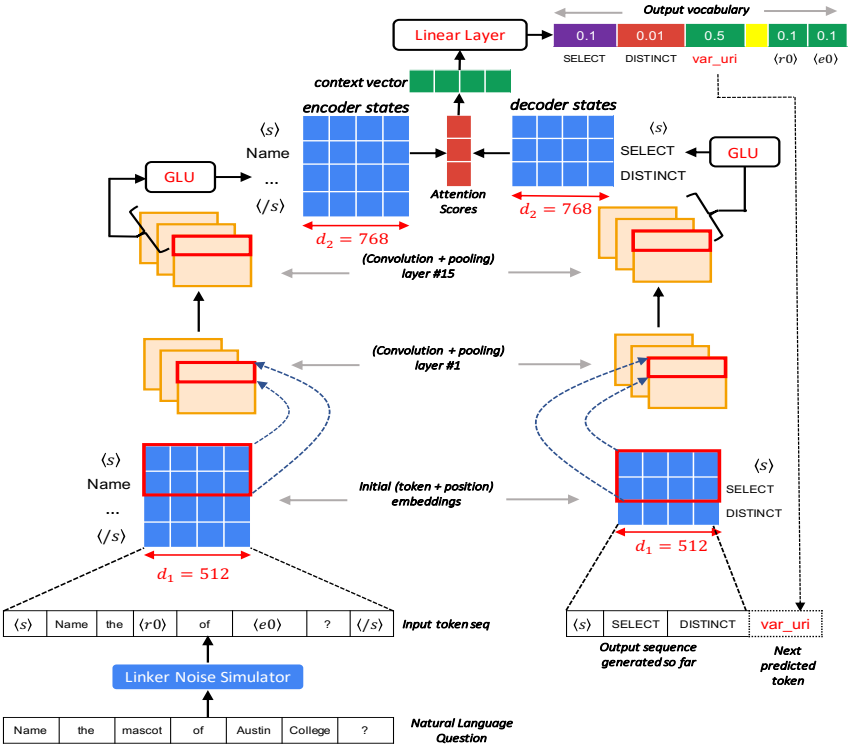


Fig. 6: A CNN-based Seq2Seq model for KGQA. The example here uses noise-free linking scenario.

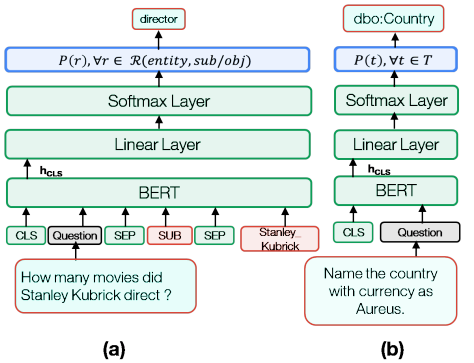


Fig. 7: Architecture of neural graph search module. (a) Relation Classifier. This module predicts relation for a given entity (b) Ontology Type Classifier. This module predicts *rdf:type* ontology class.

$(1-\gamma)/(Z-1)$ o/w, where $\gamma \in [0,1]$, $\gamma > 1/Z$.

## V. Stage-II: Neural Graph Search Module

Our error analysis on output of Stage-I revealed that entity linking performance is reasonably good but the same is not true for relation linking in case of realistic scenario. Existing literature [44], [45] also show enough evidences of achieving high performance on the entity linking task, whereas relation linking turns out to be harder due to the complexity of natural language. Because of this, we have most of the entities within a SPARQL silhouette generated by Stage-I as correct but the relations are incorrect. This motivated us to design our proposed *neural graph search* (NGS) module. The NGS module in Stage-II takes a SPARQL silhouette as input and produces an improved version of SPARQL by replacing incorrect relations. This is a BERT-based module and its architecture is shown in Figure 7. This module works as follows:

1) We consider each triple $\langle e_s, r, e_o \rangle$ in the SPARQL silhouette in which at least one of the entities is an existential variable unless the silhouette is with *rdf:type* relation which we handle separately. We prepare input in the following format: [CLS] $Q$ [SEP] [SUB (or OBJ)] [SEP] $e_s$ (or $e_o$). Here, $Q$ is the token sequence of input question text and [SUB (or OBJ)] is a special token depending on whether the grounded entity is in subject (or object) position (refer Figure 7a). We also pass grounded entity ($e_s$ or $e_o$) as the last element of this input. [CLS] and [SEP] are special tokens from BERT vocabulary.
2) We feed above input sequence of tokens into the BERT layer of graph search module. The output from the [CLS] token, $h_{CLS}$ is passed through a linear layer followed by a *softmax* layer. This softmax layer induces a probability score $p_r$ for each relation $r \in \mathcal{R}$ in the given KG. While training, we use the following loss function (given for single example): $\ell = (1-\alpha) * (\ell_c) + (\alpha) * (\ell_{gs})$. Here, $\ell_c$ denotes standard cross entropy loss between predicted probabilities $\{p_r\}_{r \in \mathcal{R}}$ and the gold relation. The graph search loss term $\ell_{gs}$ forces the predicted probabilities to be low for all those relations which are invalid relations (in the given KG) for corresponding input entity $e_s$ (or $e_o$) in the input position (subject or object). $\ell_{gs}$ is the Binary Cross-Entropy for logits followed by a sigmoid layer. For this, we assume a uniform probability distribution over all such valid relations and compute its cross entropy loss with $\{p_r\}_{r \in \mathcal{R}}$. $\alpha$ is a hyperparameter.
3) During inference, at softmax layer, we restrict the outputs only to those relations $r \in \mathcal{R}$ which are valid relations for the input entity as being subject or object. For example, if input grounded entity is $e_s$ then we restrict prediction to only those relations $r$ for which $\langle e_s, r, ?x \rangle$ is a valid triple for some grounding of $?x$. In DBpedia same relation can exist in the form of *'dbo'* and *'dbp'* for a specific entity. In such cases, we pick the *'dbo'* version as these are curated and mapped to the DBpedia ontology. Prediction is made based out of 61623 relations available in DBpedia.

4) We have a separate version of the NGS module (refer Figure 7b) if the relation $r$ in a given triple is *rdf:type*. Note, in DBpedia, a triple containing *rdf:type* relation looks like this $\langle ?x, \textit{rdf:type}, \textit{CLASS} \rangle$ where, $?x$ is a variable and *CLASS* is the DBpedia ontology class of the entity $?x$. For such triples, input to NGS module is [CLS] Q. We need to predict the corresponding ontology class, *CLASS*. DBpedia ontology contains 761 classes and hence, in this model, prediction is one of these 761 classes. This module is trained with standard cross-entropy loss. An example of the *rdf:type* classification would be to predict *dbo:Country* for the question *'Name the country with currency as Aureus?'*.

## VI. Experiments and Results

**Datasets:** We work with two different KGQA datasets based on DBpedia: LC-QuAD-1 [20] and QALD-9 [46]. We have chosen these datasets as the nature of the datasets are very different based on the way these were generated. LC-QuAD-1 is template based and contains 5000 examples.We split this dataset into 70% training, 10% validation, and 20% test sets (same as the leaderboard). QALD-9 is a multilingual, manually curated dataset. Questions in this dataset vary in terms of reasoning nature (e.g. counting, temporal, superlative, comparative, etc.) and the SPARQL aggregation functions. This dataset contains 408 training and 150 test examples. We split the training set into 90% training and 10% validation sets.

**Evaluation Metric:** Performance is evaluated based on the standard precision, recall, $F_1$ score for KGQA systems. We have used same evaluation metric as [47] and [46] for LC-QuAD-1 and QALD-9 datasets respectively. We also measure a metric called Answer Match (AM).

*Answer Match (AM)*: For a question $Q$, when executing the predicted SPARQL in the underlying KG, if we get $S_p = S_g$ then we say AM = 1 otherwise AM = 0. Here, $S_g$ and $S_p$ are gold and predicted answer set respectively.

**Baselines:** We compare our approach with three baselines: WDAqua [48], QAmp [47] and gAnswer [49], as these are the only systems which consider equivalent setting like the present work; realistic linking scenario and report accuracy numbers on the whole test set of the corresponding datasets. WDAqua is a graph based approach to generate SPARQL based on predefined patterns. These SPARQL candidates are then ranked. QAmp uses text similarity and graph structure based on an unsupervised message-passing algorithm. gAnswer which is very specific to the QALD-9 dataset, is a graph data driven approach and generates query graph to represent user intention. WDAqua and QAmp are top entries in the LC-QuAD-1 leaderboard [2] whereas, WDAqua and gAnswer in the QALD-9 challenge [46].

**Experimental Setup:** For entity/relation linking we use *Falcon* [50] since this suits our requirements the best. We use *fairseq*[3] library for implementation of the CNN-based seq2seq model [41] comprising of 15 layers.We use Nesterov Accelerated Gradient (NAG) optimizer. We experimented with different values of hyperparameters and report results on test set for the values yielding the best performance on the validation set. Optimal values are as follows: learning rate - 0.25 and $10^{-5}$ in Stage-I and Stage-II respectively, batch size - 8 for both the stages, hyperparameter $\alpha$ in Stage-II - $4 \times 10^{-1}$ and $6 \times 10^{-1}$ for the LC-QuAD-1 and QALD-9 datasets respectively. For neural graph search module, we work with a pre-trained *BERT-base uncased model*. It consists of 12 transformer layers, 12 self-attention heads, and a hidden dimension of 768. We use 2 Tesla v100 GPU for training our model.

[2]http://lc-quad.sda.tech/lcquad1.0.html

[3]https://github.com/pytorch/fairseq

TABLE II: Test set performance on LC-QuAD-1 dataset.

| Model Type | Model Name | AM | Prec. | Recall | $F_1$ |
|---|---|---|---|---|---|
| Baseline | WDAqua | - | 22.00 | 38.00 | 28.00 |
| | QAmp | - | 25.00 | **50.00** | 33.33 |
| Stage-I (Ours) | No Noise | 82.88 | 83.11 | 83.04 | 83.08 |
| | Part Noise | 41.34 | 42.40 | 42.26 | 42.33 |
| | Full Noise | 24.92 | **25.54** | 25.64 | 25.59 |
| Stage-I + Stage-II) | w/o type | 30.63 | **32.17** | 32.20 | 32.18 |
| | w/ type | 34.83 | **37.03** | 37.06 | **37.05** |

**Results and Discussions:** Table II and III capture the performance of our approach in comparison with the state-of-the-art on the LC-QuAD-1 and QALD-9 datasets. Our approach achieves state-of-the-art performance in case of LC-QuAD-1 dataset by an absolute margin of 3.72% F1. Our seq2seq model can achieve upto 83.08% $F_1$ for LC-QuAD-1 and 55.3% Macro $F_1$ QALD for QALD-9 dataset if the entity/relation linker were to be 100% correct. The gap between the performance of *No Noise* linking (upper bound) and *Full Noise* linking (lower bound) illustrates how the performance of the entity/relation linker impacts the overall performance of KGQA. Poor performance of *Falcon* on relation linking (44.99% and 37.17% of Recall on test set of the LC-QuAD-1 and QALD-9 datasets respectively) also justifies this gap. Furthermore, the performance of (Stage-I + Stage-II) demonstrates we gain 11.46% in $F_1$ and 4.2% in answer match (AM) by absolute margin for LC-QuAD-1 and QALD-9 respectively compared to the full noise scenario of Stage-I. Our results in the last two rows of both the tables (Stage-I + Stage-II) are under full noise setting for entity/relation linking. Despite of very less training data and presence of specific type of queries in QALD-9 dataset, our purely neural model is able to outperform the prior methods with substantial improvement in terms of Macro Precision.

By manually analysing examples, we find that our method can learn complex patterns (Table IV) of queries including imperative, interrogative, questions with count (aggregation) and involving multiple relations. The proposed model is a simple neural approach which does not need question interpretation step unlike QAmp and also can learn complex templates

TABLE III: Test set performance on QALD-9 dataset. Here Mac. means Macro and Rec. means Recall.

| Model Type | Model Name | AM | Mac. Prec. | Mac. Rec. | Mac. $F_1$ QALD |
|---|---|---|---|---|---|
| Baseline | WDAqua | - | 26.1 | 26.7 | 28.9 |
| | gAnswer | - | 29.3 | **32.7** | **43.0** |
| Stage-I (Ours) | No Noise | 29.9 | 80.4 | 42.1 | 55.3 |
| | Part Noise | 13.1 | 63.9 | 28.7 | 39.6 |
| | Full Noise | 11.1 | **82.6** | 23.0 | 36.0 |
| Stage-I + Stage-II) | w/o type | 15.3 | **59.4** | 26.1 | 36.2 |
| | w/ type | 15.3 | **59.4** | 26.1 | 36.2 |

without the need of a large collection of templates unlike WDAqua. As our seq2seq model translates natural language query to SPARQL, it does not need to resolve any ambiguity in the natural language questions to reach to answer unlike approaches (gAnswer) that generate query graphs. Another advantage of our method is it is KG agnostic.

**Error Analysis:** We randomly picked 50 examples where our model predictions are wrong. We see that, there are four types of scenarios where our model fails to generate correct SPARQL: (i) two very similar looking relations (refer Table V)(ii) inconsistencies in KG (iii) gold SPARQL comprising infrequent SPARQL keywords (iv) classes of rdf:type belong to classes other than DBPedia classes. The performance numbers in the last two rows of Table III are same because in the dataset there are only two such gold examples with rdf:type classes with *YAGO* ontology that our model does not support. Further analysis shows that the reason for QALD-9 having low upper bound is its training set size being too small (408) and a large variety of SPARQL keywords within a small training set. There are only 32 queries with FILTER and 4 queries with GROUP BY keyword in the training set of 408 to represent comparative/superlative questions which is too small for any neural model to learn from. Training on more data can further improve the performance. Because of the inconsistencies in KG, presence of classes in *YAGO* ontology and infrequent SPARQL keywords, generated SPARQL silhouette in QALD-9 dataset has errors other than incorrect entity/relations. Therefore, Stage-II offers much smaller gain for QALD-9. Combining LC-QuAD-1 training data with QALD-9 did not improve the performance of the QALD-9 dataset (29.60% F1 score compared to the 36.0% when trained on same domain) because the nature of the SPARQL is very different in both the datasets. Our transfer learning and supporting results with other details are provided here.[4]

## VII. Conclusion

We have proposed a simple sequential two-stage NMT-based approach to solve the KGQA task. Stage-I translates natural language query to SPARQL silhouette. To train seq2seq module, we introduced various noise scenarios with masking schemes to handle unseen entities/relations and reduce the vocabulary size. We also introduce Neural Graph Search Module in Stage-II to improve the quality of SPARQL silhouette generated in the realistic scenario at Stage-I. We demonstrate that integrating Stage-II module with Stage-I improves the overall performance of the system in the realistic scenario to achieve state-of-the-art performance or comparable results. We believe, this research demonstrates great potential of NMT-based approaches to solve the KGQA task and opens up a new research direction. In future, we plan to explore an iterative graph search approach where entity linking performance is also low.

[4]https://www.dropbox.com/sh/knur0yq1x58lrma/AADz1BaxPyDciBKWzIO2qdRqa?dl=0

TABLE IV: Anecdotal examples where our model predictions are correct.

| Question | Predicted SPARQL |
|---|---|
| Is Peter Piper Pizza in the pizza industry? | ASK WHERE { dbr:Peter_Piper_Pizza dbo:industry dbr:Pizza } |
| Which governor of Winston bryant is also the president of Carl Stokes? | SELECT DISTINCT ?uri WHERE { dbr:Winston_Bryant dbp:governor ?uri dbr:Carl_Stokes dbp:president ?uri } |
| How many other home stadium are there of the soccer club whose home stadium is Luzhniki Stadium? | SELECT DISTINCT COUNT(?uri) WHERE { ?x dbp:homeStadium dbr:Luzhniki_Stadium . ?x dbo:homeStadium ?uri } |

TABLE V: Examples of relations where our model struggles to disambiguate.

| Gold Relation | Predicted Relation |
|---|---|
| placeOfDeath | deathPlace |
| mouthPlace | sourceRegion |
| associatedBand | associatedMusicalArtist |
| product | products |


## Acknowledgement

We would like to acknowledge IBM Cognitive Computing Cluster (CCC) for providing resources to carry out various experiments.


## References


[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proc. of ACM SIGMOD*, pages 1247–1250, 2008.

[2] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia – A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6:167–195, 2015.

[3] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. YAGO 4: A reason-able knowledge base. In *Proc. of ESWC*, pages 583–596, 2020.

[4] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proc. of AAAI*, 2015.

[5] Thomas Steiner, Ruben Verborgh, Raphaël Troncy, Joaquim Gabarro, and Rik Van de Walle. Adding realtime coverage to the google knowledge graph. In *Proc. of ISWC*, 2012.

[6] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proc. of ACL*, pages 1415–1425, 2014.

[7] Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

[8] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proc. of ACL*, pages 33–43, 2016.

[9] Percy Liang. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*, 2013.

[10] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proc. of UAI*, 2005.

[11] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Proc. of ACL*, pages 956–966, 2014.

[12] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Question answering over freebase with multi-column convolutional neural networks. In *Proc. of ACL*, pages 260–269, 2015.

[13] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.

[14] Kuldeep Singh, Andreas Both, Arun Sethupat, and Saeedeh Shekarpour. Frankenstein: A platform enabling reuse of question answering components. In *Proc. of ESWC*, pages 624–638. Springer, 2018.

[15] Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. Question answering over knowledge bases by leveraging semantic parsing and neuro-symbolic reasoning. *arXiv preprint arXiv:2012.01707*, 2020.

[16] Shiqi Liang, Kurt Stockinger, Tarcisio Mendes de Farias, Maria Anisimova, and Manuel Gil. Querying knowledge graphs in natural language. *Journal of Big Data*, 8(1):1–23, 2021.

[17] Michael Petrochuk and Luke Zettlemoyer. SimpleQuestions Nearly Solved: A New Upperbound and Baseline Approach. In *Proc. of EMNLP*, pages 554–558, 2018.

[18] Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *arXiv:1607.06275*, 2016.

[19] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 7th open challenge on question answering over linked data (QALD-7). In *Semantic Web Evaluation Challenge*, pages 59–69, 2017.

[20] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. LC-QuAD: A corpus for complex question answering over knowledge graphs. In *Proc. of ISWC*, pages 210–218, 2017.

[21] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*, 2015.

[22] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to sparql. *Future Generation Computer Systems*, 117:510–519, 2021.

[23] Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. An encoder-decoder framework translating natural language to database queries. *arXiv preprint arXiv:1711.06061*, 2017.

[24] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. A survey on complex knowledge base question answering: Methods, challenges and solutions. In Zhi-Hua Zhou, editor, *Proc. of IJCAI*, pages 4483–4491, 2021. Survey Track.

[25] Luke Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proc. of EMNLP*, pages 678–687, 2007.

[26] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proc. of EMNLP*, pages 1533–1544, 2013.

[27] Dennis Diefenbach, Kamal Singh, and Pierre Maret. Wdaqua-core0: A question answering component for the research community. In *Semantic Web Evaluation Challenge*, pages 84–89, 2017.

[28] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. Constraint-based question answering with knowledge graph. In *Proc. of COLING*, pages 2503–2514, 2016.

[29] Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. On generating characteristic-rich question sets for qa evaluation. In *Proc. of EMNLP*, pages 562–572, 2016.

[30] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *Proc. of ISWC*, pages 69–78, 2019.

[31] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proc. of ACL*, pages 1321–1331, 2015.

[32] Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. Learning to rank query graphs for complex question answering over knowledge graphs. In *Proc. of ISWC*, pages 487–504, 2019.

[33] Jiwei Ding, Wei Hu, Qixin Xu, and Yuzhong Qu. Leveraging frequent query substructures to generate formal queries for complex question answering. In *Proc. of EMNLP*, pages 2614–2622, 2019.

[34] Yunshi Lan and Jing Jiang. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proc. of ACL*, pages 969–974, 2020.

[35] Yongrui Chen, Huiying Li, Yuncheng Hua, and Guilin Qi. Formal query building with query structure prediction for complex question answering over knowledge base. In *Proc. of IJCAI*, pages 3751–3758, 2020.

[36] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, 2017.

[37] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir R. Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proc. of NAACL-HLT*, 2018.

[38] Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. An encoder-decoder framework translating natural language to database queries. In *Proc. of IJCAI*, pages 3977–3983, 2018.

[39] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proc. of ACL*, pages 440–450, 2017.

[40] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[41] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proc. of ICML*, 2017.

[42] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proc. of ICML*, pages 933–941, 2017.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, pages 770–778, 2016.

[44] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. Scalable zero-shot entity linking with dense entity retrieval. In *Proc. of EMNLP*, November 2020.

[45] Belinda Z. Li, Sewon Min, Srini Iyer, Yashar Mehdad, and Wen tau Yih. Efficient one-pass end-to-end entity linking for questions. In *Proc. of EMNLP*, November 2020.

[46] Ngonga Ngomo. 9th challenge on question answering over linked data (qald-9). *language*, 7(1):58–64, 2018.

[47] Svitlana Vakulenko, Javier David Fernandez Garcia, Axel Polleres, Maarten de Rijke, and Michael Cochez. Message passing for complex question answering over knowledge graphs. In *Proc. of CIKM*, pages 1431–1440, 2019.

[48] Dennis Diefenbach, Andreas Both, Kamal Deep Singh, and Pierre Maret. Towards a question answering system over the semantic web. *Semantic Web*, pages 421–439, 2020.

[49] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over rdf: a graph data driven approach. In *Proc. of ACM SIGMOD*, pages 313–324, 2014.

[50] Ahmad Sakor, Kuldeep Singh, and M E Vidal. Falcon: An entity and relation linking framework over DBpedia. In *Proc. of CEUR Workshop*, volume 2456, pages 265–268, 2019.