Less is More: Data-Efficient Complex Question Answering over Knowledge Bases

Yuncheng Hua[a,d], Yuan-Fang Li[b], Guilin Qi[a,c,*], Wei Wu[a], Jingyao Zhang[a], Daiqing Qi[a]

[a] *School of Computer Science and Engineering, Southeast University, Nanjing, China*
[b] *Faculty of Information Technology, Monash University, Melbourne, Australia*
[c] *Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China*
[d] *Southeast University-Monash University Joint Research Institute, Suzhou, China*

Abstract

Question answering is an effective method for obtaining information from knowledge bases (KB). In this paper, we propose the Neural-Symbolic Complex Question Answering (NS-CQA) model, a data-efficient reinforcement learning framework for complex question answering by using only a modest number of training samples. Our framework consists of a neural *generator* and a symbolic *executor* that, respectively, transforms a natural-language question into a sequence of primitive actions, and executes them over the knowledge base to compute the answer. We carefully formulate a set of primitive symbolic actions that allows us to not only simplify our neural network design but also accelerate model convergence. To reduce search space, we employ the copy and masking mechanisms in our encoder-decoder architecture to drastically reduce the decoder output vocabulary and improve model generalizability. We equip our model with a memory buffer that stores high-reward promising programs. Besides, we propose an adaptive reward function. By comparing the generated trial with the trials stored in the memory buffer, we derive the curriculum-guided reward bonus, i.e., the proximity and the novelty. To mitigate the sparse reward problem, we combine the adaptive reward and the reward bonus, reshaping the sparse reward into dense feedback. Also, we encourage the model to generate new trials to avoid imitating the spurious trials while making the model remember the past high-reward trials to improve data efficiency. Our NS-CQA model is evaluated on two datasets: CQA, a recent large-scale complex question answering dataset, and WebQuestionsSP, a multi-hop question answering dataset. On both datasets, our model outperforms the state-of-the-art models. Notably, on CQA, NS-CQA performs well on questions with higher complexity, while only using approximately 1% of the total training samples.

Keywords: Knowledge Base, Complex Question Answering, Data-efficient, Neural-symbolic Model, Reinforcement learning

1. Introduction

Knowledge base question answering (KBQA) [1, 2, 3, 4, 5] is an active research area that has attracted significant attention. KBQA aims at interpreting natural-language questions as logical forms, action sequences, or programs, which could be directly executed on a knowledge base (KB) to yield the answers.

Many techniques have been proposed for answering single-hop or multi-hop questions over a knowledge base. Neural network-based methods [1, 2, 3, 4, 5] represent the state-of-the-art in KBQA. More recently, complex knowledge base question answering (CQA) [6] has been proposed as a more challenging task. Complex question answering, the subject of this paper, focuses on aggregation and multi-hop questions, in which a sequence of discrete operations – e.g., set conjunction, counting, comparison, intersection, and union – needs to be executed to derive the answer.

Complex question answering is typically cast as a *semantic parsing* problem, whereby natural-language questions are transformed into appropriate structural queries (sequences of discrete actions). Such queries are then executed on the knowledge base to compute the answer. Consider the complex question "How many rivers flow through India **and** China?" as a motivating example. Fig. 1 shows an incomplete sub-graph

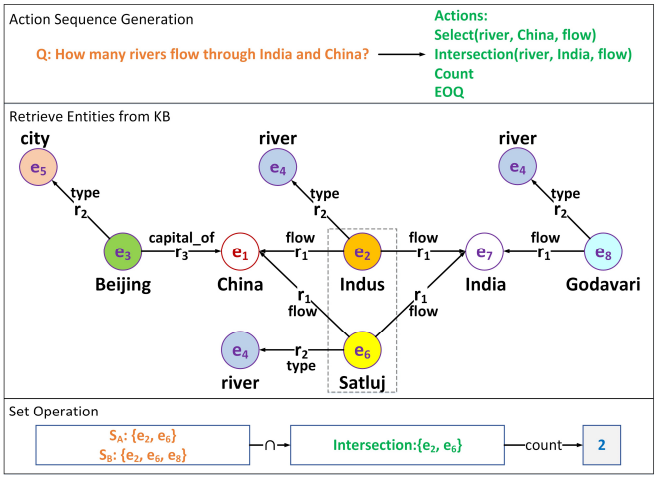

Figure 1: An example illustrating the task of complex question answering.

*Corresponding author
Email addresses: devinhua@seu.edu.cn (Yuncheng Hua), yuanfang.li@monash.edu (Yuan-Fang Li), gqi@seu.edu.cn (Guilin Qi), wuwei@seu.edu.cn (Wei Wu), zjyao@seu.edu.cn (Jingyao Zhang), daiqing_qi@seu.edu.cn (Daiqing Qi)

relevant to this question. To answer this question, all entities whose type is "river" and link to the entity "China" with edge "flow" will first need to be retrieved from the KB to form the candidate set S_A. Meanwhile, the candidate set S_B will also be formed to represent those rivers that flow through India. After obtaining the intersection of S_A and S_B, the number of elements in the intersection can finally be identified as the correct answer to the question. It can be seen that a diverse set of operations, including selection, intersection, and counting operations need to be sequentially predicted and executed on the KB.

Sequence-to-sequence (seq2seq) models learn to map natural language utterances to executable programs, and are thus good model choices for the complex question answering task. However, under the supervised training setup, such models require substantial amounts of annotations, i.e., manually annotated programs, to effectively train. For practical KBQA applications, gold annotated programs are expensive to obtain, thus most of the complex questions are not paired with the annotations [7]. Reinforcement learning (RL) is an effective method for training KBQA models [8, 9] as it does not require annotations, but only denotatinons (i.e. answers) as weak supervision signals. However, RL-based KBQA methods face a number of significant challenges.

Sparse reward. Neural-symbolic models that are optimized through RL have been proposed for the complex question answering task [8, 7, 9]. In the RL context, questions of the same pattern could be regarded as one single *task*, while programs trying to solve these similar questions are considered *trials*. Instead of using the gold annotations, neural-symbolic models employ *rewards*, i.e., comparisons between the predicted answer and the ground-truth answer as the distant supervision signal to train the policy [8, 9]. Usually, a positive reward could only be given at the end of a long sequence of correct actions. However, in the initial stage of model training, most of the trials sampled from the sub-optimal policy attain small or zero rewards [10]. We randomly selected 100 questions from a complex question answering dataset, and manually inspect the generated trials. We found that more than 96.5% of the generated trials led to wrong answers and got zero reward. Thus, this **sparse reward** problem in the complex question answering task is a major challenge that current neural-symbolic models face.

Complex Imperative Program Induction from Terminal Rewards (CIPITR) [7] is the state-of-the-art method for the complex question answering task. It employs high-level constraints and additional auxiliary rewards to alleviate the sparse reward problem. By using pre-defined high-level constraints, CIPITR restricts the model to search for possible actions that are semantically correct. Besides, it rewards the model by the additional feedback when the generated answers have the same type of ground-truth answers. However, on the one hand, defining the high-level constraints comes at the cost of manual analysis to guide the model's decoding process, which is tedious and expensive. On the other hand, CIPITR only harnesses the type of predicted answers as the auxiliary reward, which makes many failed experience still only have a zero reward. At the initial stage of training, most of the programs generated by CIPITR fail to yield expected answers and gain zero rewards, thus most of the programs do not contribute to model optimization. Thus, the sparse reward problem remains a challenge for CIPITR.

Data inefficiency. Furthermore, due to the sparse supervision signals, such models are often **data inefficient**, which means many trials are required to solve a particular task [11]. Being trained from scratch, RL models often need thousands of trials to learn a simple task, no matter what policy is employed to search programs. When faced with a large number of questions, training such models would consume an enormous amount of time. One way to increase data efficiency is to acquire task-related prior knowledge to constrain the search space. However, in most cases, such prior knowledge is unavailable unless manual labeling is employed. Hence, the data inefficiency problem often makes models expensive to train and thus infeasible/impractical. Liang et al [8] proposed the Neural Symbolic Machine (NSM) that maintains and replays *one* pseudo-gold trial that yields the highest reward for each training sample. When using RL to optimize the policy, NSM assigns a deterministic probability to the best trial found so far to improve the training data efficiency. However, in NSM, the best trial to be replayed might be a spurious program, i.e., an incorrect program that happens to output the correct answer. Under such circumstances, NSM would be misguided by the spurious programs since such programs could not be generalized to other questions of the same pattern. Besides, NSM only harnesses the accuracy of the predicted answers to measure the reward, hence also suffers from the sparse reward problem.

Large search space. Large KBs, like Wikidata [12] or Freebase [13], contain comprehensive knowledge and are suitable to be sources for complex question answering. The KBs with smaller size usually embrace a limited number of facts, thus are insufficient to yield the required answers. To answer the questions, searching for KB artifacts (entities, classes, and predicates in KB) that are related to the questions is a crucial step. However, large KBs often lead to vast search space. Given a sequence of actions, at each step of its execution, both the operator and the parameters, i.e., KB artifact, used in action need to be correct. To produce the correct result, the actions in the sequence also need to follow a particular order. With the above factors combined, searching desired action sequences would consume a considerable amount of time and memory, which makes the training process slow and expensive. How to design a set of simple yet effective actions that could reduce the search space remains a challenge.

In this paper, we propose a Neural-Symbolic Complex Question Answering framework, NS-CQA. It trains a policy to generate the desired programs from the comparison between the generated answer and the ground-truth result. We incorporate a memory buffer, design an adaptive reward function, and propose a curriculum-guided reward bonus to improve the model.

To **reduce search space**, we use a combination of simple yet effective techniques to reduce model complexity, increase generalizability, and expedite training convergence. We employ the widely-used masking technique to allow the model to handle unseen KB artifacts effectively. In conjunction, the copy mechanism [14] is also employed to reduce the size of the decoder

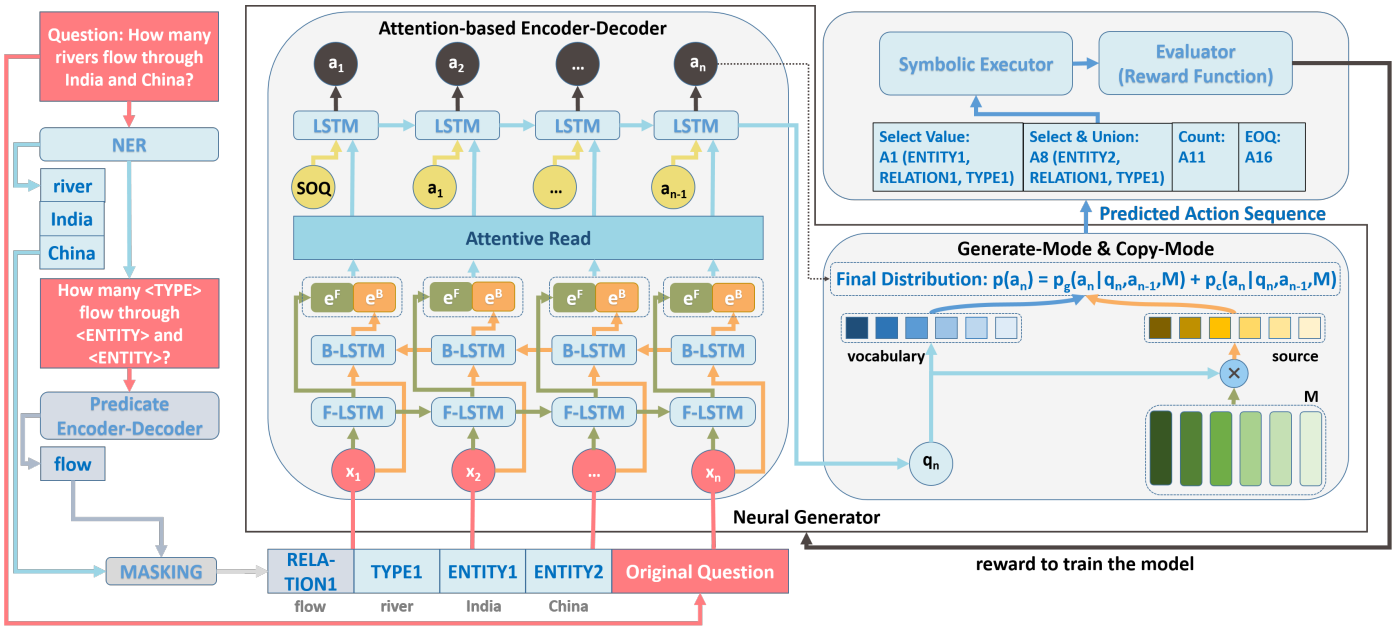

Figure 2: Overview of the proposed NS-CQA model.

output vocabulary drastically. Accordingly, the decoder output vocabulary only needs to contain the primitive actions and a handful of masks, instead of the entire KB. Also, we carefully craft a set of primitive actions that are necessary for the complex question answering task and simplify the query form to reduce the search space. Our primitive actions free the model from the need for maintaining expensive and sophisticated memory modules. On the contrary, the model maintains the intermediate results in a simple key-value dictionary. The model can directly compute the final answer to a question by executing a correct sequence of primitive actions.

Different from previous neural-symbolic models for the complex question answering task, our NS-CQA framework is designed to **augment the sparse extrinsic reward** by a dense intrinsic reward and exploit trials efficiently.

Instead of using the manually defined high-level constraints as in CIPITR, we employ a random search algorithm to find a few pseudo-gold annotations that lead to correct answers. The pseudo-gold annotations are used as supervision signals to pre-train the model, which could guide the model to filter out infeasible action sequences. Consequently, in the initial training stage, we mitigate the cold start issue by using the pseudo-gold annotations as demonstration data to pre-train our model.

Moreover, instead of recording one trial for each question, we maintain a memory buffer to store the promising trials, i.e., the action sequences that lead to the correct answer or gain high rewards. In our work, on the one hand, we aim to converge a behavior policy to a target optimal policy. Thus we need to measure how similar/important the generated trials are to trials that the target policy may have made. We design a reward bonus, **proximity**, to favor these "similar/important" trials. On the other hand, to avoid overfitting the spurious trials, we also encourage the policy to explore in undiscovered search space, i.e., the space that is beyond the imitation of the pseudo-gold annotations. Therefore, we design another reward bonus, **novelty**, to the generated action sequences that are "different" from the trials stored in the memory buffer.

To adaptively control the exploration-exploitation trade-off, we employ a *curriculum learning* [15] scheme to dynamically change the influence of the two reward bonuses, namely the proximity and the novelty. Particularly, given a question, we define the proximity for the predicted trial as the highest similarity between the predicted trial and all the trials in the memory buffer. Novelty is also defined through similarity. We consider a trial is novel if it is not similar to all the trials in the memory buffer. At the initial stage, since the policy is sub-optimal and the trials generated by the policy are generally infeasible, we prefer more novelty for exploration. We gradually increase the proportion of proximity and reduce the influence of novelty during the later training epochs. At the final stage of the training, the proportion of proximity in the reward bonus will rise to 100%. Such a delicately designed curriculum method can significantly improve the learning quality and efficiency [16].

Besides, to alleviate the sparse reward problem, an adaptive reward function (ARF) is proposed. ARF encourages the model with partial reward and adapts the reward computing to different question types. We sum up the reward bonus with the adaptive reward to make our RL model learn from the combined reward. With this modification, we reshape the sparse rewards into dense rewards and enable any failed experience to have a nonnegative reward, thus alleviate the sparse reward problem.

Furthermore, by incorporating a memory buffer, which maintains off-policy samples, into the policy gradient framework, we **improve the sample efficiency** of the REINFORCE algorithm in our work [11]. We encourage the model to generate trials that are similar to the trials in the memory buffer by using the proximity bonus. Therefore the high-reward trials could be re-sampled frequently to avoid being forgotten in the training

process. Since a group of question shares the same pattern, a sequence of the same actions could solve such questions. Once we improve sample efficiency, we reduce the trials needed for training. Therefore, using a minimal subset of training samples, our model can produce competitive results.

Overall, the main contributions in this paper can be summarized as follows.

1. A neural-symbolic approach that is augmented by memory buffer and is designed for complex question answering. In our method, for each question, we resort to the previous promising trials stored in the memory to assist our model in replaying and generating feasible action sequences.
2. A curriculum-learning scheme that adaptively combines novelty and proximity to balance the exploration-exploitation trade-off for the model. We treat the combination of the novelty and proximity as a bonus to the reward, therefore alleviate the sparse reward and data inefficiency problems.
3. Several simple yet effective techniques are proposed to reduce search space, improve model generalizability, and accelerate convergence. In our work, the masking method and the copy mechanism are incorporated in Seq2Seq learning to avoid searching over a large action space. Also, a set of primitive actions is carefully designed to solve complex questions by executing actions sequentially, avoiding the maintenance of complex memory modules.

Our experiments on a large complex question answering dataset [6] and a relatively smaller multi-hop dataset WebQuestionsSP [17] show that NS-CQA outperforms all the recent, state-of-the-art models. Moreover, NS-CQA performs well on the questions with higher complexity, demonstrating the effectiveness of our method.

The rest of this paper is organized as follows. Related works are introduced in Section 2. Our NS-CQA framework is described in Section 3. Section 4 describes the experiments and evaluation results. In Section 5, we perform some qualitative analysis, showing positive examples and typical errors. We conclude our work in Section 6.

2. Related work

The NS-CQA model is inspired by two lines of work: semantic parsing and neural-symbolic systems. Semantic parsing mainly focuses on reformulating natural language questions into logic forms, which are then executed on knowledge bases (KBs) to compute answers [18, 19, 20]. More recent approaches employ sophisticated deep learning models to search entities and predicates that are most relevant to the question [21, 22, 23].

Many of these works tackle simple one-hop questions that are answerable by a single triple. Others address the multi-hop task, in which answers are entities that can be retrieved by a path of connected triples. In both cases, a model only retrieves entities as answers in a fixed search space, i.e., the KB. Our model addresses a more challenging problem, where answers come from a much larger search space, involving not only entities in the KB but also their logical combinations and aggregations (numbers).

The complex KBQA dataset, such as CQA [6], requires the execution of discrete actions rather than merely searching for entities in KB. Memory network [4, 24, 25, 26] is used to store facts in KB and makes it possible to solve complex questions. Luo et al. [27] encodes complex questions into vectors to represent the semantics and structure of the input sentence contemporaneously, by which the similarity between the question and the graph could be computed. Dialog-to-Action (D2A) [28] incorporates dialog memory management in generating logical forms that would be executed on a large KBs to answer complex questions. It labels all training samples with pseudo-gold actions and trains the model by imitation learning. It is worth noting that D2A aims to answer *context-dependent* questions, where each question is part of a multiple-round conversation. On the other hand, in this paper, we consider answering the single-turn questions. Therefore we do not include D2A as a baseline method in the evaluation as it is not directly comparable to our problem setup. The semantic parsing approaches mentioned so far all require the annotation of the entire training dataset to learn the models. Different from these approaches, our model can be learned from a small number of pseudo-gold annotations only.

Some recent studies in KBQA focus on semi-supervised learning, in which models are trained solely on denotations, e.g., the execution results of queries, the state of the final time step, etc. In [1], a semantic parser is trained by learning from question-answer pairs rather than annotated logical forms to query the knowledge graph. Furthermore, the parser could be trained without manual annotations or question-answer pairs by treating denotations of natural language questions and related KB queries as weak-supervision [29]. Similarly, queries automatically generated from knowledge graph triples and paraphrased questions without answers are used in weak-supervision to train subgraph embedding models [30]. To fully understand the intention of questions, STAGG [5] is proposed to generate a staged query graph and directly map the graph into λ-calculus.

Neural-symbolic models integrate neural networks with logic-based symbolic executors to conduct non-differentiable computations. Neural Turing Machines (NTMs) [31] pioneer the neural-symbolic methods, in which REINFORCE is employed to train the model in the RL Neural Turing Machines (RL-NTMs) [32]. When answering natural language questions on relational tables, some approaches [33, 34] predict discrete symbolic operations by neural networks and obtain answers by executing them. Guu et al. [35] marry RL and maximum marginal likelihood (MML) to avert the spurious problems. The neural-symbolic visual question answering (NS-VQA) system [36] combines deep representation learning and symbolic program execution to solve visual question answering problems over a synthetic image dataset.

Most relevant to this work are two state-of-the-art techniques on complex KBQA: Neural Symbolic Machines (NSM) [8] and CIPITR [37]. NSM deals with multi-hop questions in the

WebQuestionsSP dataset [17] with two components: the programmer and the computer. By employing an EM-like mechanism, NSM iteratively finds the pseudo-gold trials for the training questions. NSM then assigns the pseudo-gold trials with a deterministic probability, therefore, to anchor the model to the high-reward trials. In a similar vein, CIPITR translates a natural-language complex question into a multi-step executable program using the Neural Program Induction (NPI) technique. CIPITR does not require gold annotations and can learn from auxiliary rewards, KB schema, and inferred answer types.

Our neural-symbolic model is different from them in that we augment our RL-based model with a memory buffer to record the successful trials. With the help of the memory buffer, we could compute the extra reward bonus to encourage the model to generate new trials, imitate successful trials, and reshape the sparse reward to provide dense feedback. As can be seen in Table 3, our NS-CQA model outperforms all the baseline models, and the performance difference is prominent on the more complex categories of questions. Also, in Table 4, we can find that NS-CQA is better than other baseline models. The superiority of NS-CQA in both the datasets verified the effectiveness and generalization ability of the model.

3. NS-CQA: A Complex Question Answering Approach

In our work, the complex question answering problem is regarded as a semantic parsing task: given a complex question q consisting of tokens $(w_1, \ldots, w_m)$, the model generates a primitive action sequence $(a_1, \ldots, a_q)$, and execute the sequence on the KB $\mathcal{K}$ to yield the answer a.

This section outlines our NS-CQA approach to the complex question answering problem. We first describe the set of primitive actions in Section 3.1. Given a complex question, the *semantic parser* (Section 3.2) recognizes KB artifacts that are relevant to the question. By combining the question and the output of the parser, the *neural generator* (Section 3.3) transforms the query into a sequence of primitive actions.

The *symbolic executor* (Section 3.4) executes the actions on the KB to obtain an answer. Overall, we employ *RL* to directly optimize the generator through a policy gradient on the answer predicted by the executor (Section 3.5). The high-level architecture of our model is depicted in Fig. 2.

3.1. Primitive Actions

We propose a set of primitive actions based on the subset of SPARQL queries that are necessary for the current complex question answering task and simplify the query form to reduce the search space. Our actions are designed to be simple, dispensing with SPARQL features, including namespaces, etc. It also does not support certain SPARQL features, including OPTIONAL, FILTER, etc. Since our actions belong to a subset of SPARQL's operators, the complexity of our actions follows that of SPARQL. The main contributions of this paper relate to the neural generator of these programs. Thus we leave the study of operator complexity to future research.

Unlike NSM [8], which introduces the variables to save the intermediate results, we employ a **key-value memory** to maintain the intermediate result. NSM adds a new intermediate variable into the decoder vocabulary after an action is executed, thus enables the decoder to generate the new variable in later decoding steps. Consequently, NSM dynamically increases the size of the decoder vocabulary in the decoding process. On the contrary, with the help of the memory, our model does not need to refer to previous intermediate variables, thus fix the size of the decoder vocabulary to simplify the model.

We use two components, i.e., an operator and a list of variables, to compose the primitive actions. After analyzing different types of complex problems, we design 17 operators in this work, which are described in Table 2. Besides, the key-value dictionary $\mathcal{D}$ is designed to store intermediate results. Keys in $\mathcal{D}$ refer to entities present in the question or specific special symbol, and the values are the obtained elements related to it. Before the execution of the whole action sequence, $\mathcal{D}$ is initialized as empty. When executing an action, the model will generate an intermediate result based on content in $\mathcal{D}$, which is the result derived from the last action. Then the generated intermediate result will be further stored in $\mathcal{D}$ to update it. The contents of updated $\mathcal{D}$ are then preserved for later use.

For instance, when dealing with the question "Which country has *maximum number* of rivers?", the desired output actions should be "*SelectAll(country, flow, river), ArgMax, EOQ*". The first action has an operator *SelectAll*, a relation variable *flow* and two type variables, i.e., *country* and *river*, while no entity variable is found in this action. Note that the second action only has one operator *ArgMax*, and so does the third action *EOQ*. By performing the first action, we retrieve KB to find all entities belong to type 'country' as keys. Meanwhile, we set the river-type entities linked with country-type entities by relation 'flow' as values. Like what is demonstrated in Table 1 (the retrieved results presented in the table are not consistent with the actual KB while are only for demonstration), one country-type entity is *USA* where the linked river-type objects are {*Mississippi, Colorado, Rio Grande*}. The retrieved key-value pairs (the key is one country-type entity and value is a set of river-type entities) are then stored in dictionary $\mathcal{D}$ as the intermediate result of this action. After that, the second action is executed to find the key whose mapped value has most elements. In could be found in Table 1 *Russia* has most elements thus *Russia:*{*Volga, Moskva, Neva, Ob*} is then kept in $\mathcal{D}$ and other key-value pairs are removed. Then we update $\mathcal{D}$ and view this key-value pair as the intermediate result of the second action. When encountered with *EOQ*, the model outputs the final result in $\mathcal{D}$.

To simplify the action sequence, we design particular actions (GreaterThan, LessThan, Inter, Union, and Diff) as relatively '**high-level**' actions that need multiple set operations to perform. Though such design will enhance the difficulty of symbolic executor, on the other hand, it could reduce the complexity of the neural generator. Since the bottleneck of our model lies in the difficulty of training generator, we make a compromise between executor and generator.

Take the question "What rivers flow in India *but not* China?" as an example. The reference action sequence should be "*Se-*

Table 1: Demonstration of how intermediate result evolves when executing actions sequentially.

Question1: Which country has **maximum number** of rivers?	**Actions**	**Action1:** *SelectAll (country, flow, river)*	**Action2:** *ArgMax*	**Action3:** *EOQ*
	Retrieved Key-Value Pairs	{China:{Indus, Satluj}} {India:{Indus, Satluj, Godavari}} {Russia:{Volga, Moskva, Neva, Ob}} {USA:{Mississippi, Colorado, Rio Grande}}	/	/
	Dictionary	{China:{Indus, Satluj}} {India:{Indus, Satluj, Godavari}} {Russia:{Volga, Moskva, Neva,Ob}} {USA:{Mississippi, Colorado, Rio Grande}}	{Russia:{Volga, Moskva, Neva, Ob}}	{Russia:{Volga, Moskva, Neva, Ob}}
Question2: What rivers flow in India **but not** China?	**Actions**	**Action1:** *Select (India, flow, river)*	**Action2:** *Diff (China, flow, river)*	**Action3:** *EOQ*
	Retrieved Key-Value Pairs	{India:{Indus, Satluj, Godavari}}	{China:{Indus, Satluj}}	/
	Dictionary	{India:{Indus, Satluj, Godavari}}	{India:{Godavari}}	{India:{Godavari}}

lect(India, flow, river), Diff(China, flow, river), EOQ". After executing the first action, a set of rivers flowing in India is stored in $\mathcal{D}$ as the value of the key *India*. As showed in Table 1, such key-value pair is *India:*{*Indus, Satluj, Godavari*}. When executing the second action "*Diff(China, flow, river)*", all river entities linked to *China* by relation *flow* are first retrieved from the KB. Then based on the key-value pair stored in $\mathcal{D}$, the entities indexed to India but not China will be kept as the updated value of the key *India*. In this case is *India:*{*Godavari*}. Then the key-value pair in $\mathcal{D}$ is updated accordingly. Upon encountering the action *EOQ*, the key-value pair stored in $\mathcal{D}$ is returned as the final answer to this question.

As described above, the model executes all the actions in sequence. With the help of a key-value dictionary $\mathcal{D}$, the intermediate result of the current action is recorded and preserved for later use. The model could perform the following actions based on the result stored in $\mathcal{D}$, and the new result would further update $\mathcal{D}$. Therefore, the design of $\mathcal{D}$ makes executing actions sequentially possible.

3.2. Semantic Parser

Given a natural language question, the parser first recognizes entity mentions (for example *India*) and class mentions (for example *river*) [38]. A **Bidirectional-LSTM-CRF** model is employed to label the entity/type mentions [39]. The parser then links them with the corresponding entities and types in KB. At first, the parser tries to retrieve the entity/type candidates related to mentions by computing the literal similarities. Besides, the description of the candidates and the question are embedded into vectors to get semantic similarity. The literal and semantic similarities are thus integrated to rank the entity/type candidates, while the ones have the highest score is selected as linked entities/types. Subsequently, the entity/class mentions in the query are replaced with wild-card characters to generate patterns. We employ a **convolutional Seq2Seq model** [40] to transform the the generated patterns to corresponding relations.

3.3. Neural Generator

Our generator is an attention-based Seq2Seq model augmented with the copy and masking mechanisms. Given a question with tokens $(w_1, \ldots, w_m)$, the generator predicts tokens $(a_1, \ldots, a_q)$. The input of the model is the original complex question concatenated with KB artifacts generated by the semantic parser, and the output is the tokens of a sequence of actions. In our work, the output at each time step is a single token which is used to compose actions with adjacent output tokens.

For a vanilla Seq2Seq model, all the KB artifacts corresponding to all questions will need to be collected in advance to make the vocabulary large enough to cover all questions. Let N represent the maximum number of actions in sequences. Since we have designed 17 different operators in our work, and each of operators can take up to three arguments (see Table 2 for details), in the worst case, the vocabulary size of the decoder is $O(17^N * |\mathcal{E}|^{2N} * |\mathcal{P}|^N)$, where $|\mathcal{E}|$ and $|\mathcal{P}|$ denote the number of entities (including types) and predicates in the KB $\mathcal{K}$ respectively. Given a large KB such as Freebase, $|\mathcal{E}|$ and $|\mathcal{P}|$ can be very large. Such a vocabulary size would be prohibitively large

Table 2: The set of primitive actions. $\mathcal{K}$ represents the knowledge base, $e, e_1, e_2, \ldots$ represent entities, r represents a relation, t represents a type, and $\mathcal{D}$ represents a key-value dictionary which stores intermediate results.

ID	Action	Retrieved Key-Value Pairs	Output
A1	$Select(e, r, t)$	$\{e_2 \| e_2 \in t, (e, r, e_2) \in \mathcal{K}\}$	$\mathcal{D} = \mathcal{D} \cup \{e : \{e_2\}\}$
A2	$SelectAll(et, r, t)$	$\{e_2 \| e_1 \in et, e_2 \in t, (e_1, r, e_2) \in \mathcal{K}\}$	$\mathcal{D} = \mathcal{D} \cup \{e_1 : \{e_2\}\}$
A3	$Bool(e)$	$value = 1\ if\ e \in \mathcal{D}$; $otherwise\ value = 0$	$\mathcal{D} = \{bool : value\}$
A4	$ArgMin$	$\{e_1 \| e_1 \in \mathcal{D}, \exists(e_1 : \{e_2\}) \in \mathcal{D}, \forall e' : (e' : \{e'_2\}) \in \mathcal{D}, \|\{e_2\}\| \leq \|\{e'_2\}\|\}$	$\mathcal{D} = \{e_1 : \{e_2\}\}$
A5	$ArgMax$	$\{e_1 \| e_1 \in \mathcal{D}, \exists(e_1 : \{e_2\}) \in \mathcal{D}, \forall e' : (e' : \{e'_2\}) \in \mathcal{D}, \|\{e_2\}\| \geq \|\{e'_2\}\|\}$	$\mathcal{D} = \{e_1 : \{e_2\}\}$
A6	$GreaterThan(e)$	$\{e_1 \| e_1 \in \mathcal{D}, \exists(e_1 : \{e_2\}) \in \mathcal{D}, \exists e'_2 : (e : \{e'_2\}) \in \mathcal{D}, \|\{e_2\}\| \geq \|\{e'_2\}\|\}$	$\mathcal{D} = \{e_1 : \{e_2\}\}$
A7	$LessThan(e)$	$\{e_1 \| e_1 \in \mathcal{D}, \exists(e_1 : \{e_2\}) \in \mathcal{D}, \exists e'_2 : (e : \{e'_2\}) \in \mathcal{D}, \|\{e_2\}\| \leq \|\{e'_2\}\|\}$	$\mathcal{D} = \{e_1 : \{e_2\}\}$
A8	$Inter(e, r, t)$	$\{e_2 \| e_2 \in t, (e, r, e_2) \in \mathcal{K}\}$	$\mathcal{D} = \mathcal{D} \cap \{e : \{e_2\}\}$
A9	$Union(e, r, t)$	$\{e_2 \| e_2 \in t, (e, r, e_2) \in \mathcal{K}\}$	$\mathcal{D} = \mathcal{D} \cup \{e : \{e_2\}\}$
A10	$Diff(e, r, t)$	$\{e_2 \| e_2 \in t, (e, r, e_2) \in \mathcal{K}\}$	$\mathcal{D} = \mathcal{D} - \{e : \{e_2\}\}$
A11	$Count$	$Card(\mathcal{D}) = \|\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}\}\|$	$\mathcal{D} = \{num : Card(\mathcal{D})\}$
A12	$AtLeast(n)$	$\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}, \|\{e_2\}\| \geq n\}$	$\mathcal{D} = \{e : \{e_2\}\}$
A13	$AtMost(n)$	$\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}, \|\{e_2\}\| \leq n\}$	$\mathcal{D} = \{e : \{e_2\}\}$
A14	$EqualsTo(n)$	$\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}, \|\{e_2\}\| = n\}$	$\mathcal{D} = \{e : \{e_2\}\}$
A15	$GetKeys$	$\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}\}$	$\mathcal{D} = \{key : \{e\}\}$
A16	$Almost(n)$	$\{e \| e \in \mathcal{D}, \exists(e : \{e_2\}) \in \mathcal{D}, \|\|\{e_2\}\| - n\| \leqslant \alpha\}$ where α is predefined	$\mathcal{D} = \{e : \{e_2\}\}$
A17	EOQ	*end of sequence*	$\mathcal{D}$

for the decoder, making it highly unlikely to generate the correct token, thus negatively affecting the rate of convergence.

By incorporating the masking mechanism, the names of KB artifacts used for compose actions are replaced with *masks* such as <ENTITY1>, <TYPE1> and <PREDICATE1>. Thus, an action sequence consisted of N actions will be masked into the following form: $A^{(1)}(\texttt{<E}_1\texttt{>}, \texttt{<P}_1\texttt{>}, \texttt{<E}_2\texttt{>}), \ldots, A^{(N)}(\texttt{<E}_{2N-1}\texttt{>}, \texttt{<P}_N\texttt{>}, \texttt{<E}_{2N}\texttt{>})$. For instance, the action sequence '*Select(India, flow, river), Diff(China, flow, river), EOQ*' is used to solve the problem 'What rivers flow in India but not China?'. After masking, the real names of artifacts in actions are substituted with *masks*, and the action sequence is changed into '*Select(ENTITY1, PREDICATE1, TYPE1), Diff(ENTITY2, PREDICATE1, TYPE1), EOQ*'. The mappings between the actual names and *masks* are recorded in our model, which will be later used to recover the exact names of KB artifacts in actions when being executed. Given the maximum number of actions N, with the masking mechanism, the decoder vocabulary size is reduced to $O(17^N * (2N)^{2N} * N^N)$, where $(2N) \ll |\mathcal{E}|$ and $N \ll |\mathcal{P}|$. As action sequences are typically not long (i.e., $N \leq 5$ in our observation), this represents orders of magnitude reduction in vocabulary size.

Also, we could alleviate the Out Of Vocabulary (OOV) problem with the help of the masking mechanism. OOV words refer to unknown KB artifacts that appear in the testing questions but not included in the output vocabulary and would make the generated action incomplete. When facing a question with unseen KB artifacts, the model is not able to predict such objects since they are out of the output vocabulary. However, when employing the masking mechanism, all the KB artifacts are translated into masks, thus enabling the model to select masks from relatively fixed output vocabulary. Like the above example presented, the KB artifacts 'India' and 'China' are replaced with the mask 'ENTITY1' and 'ENTITY2'. Thus our model only needs to generate the masked tokens instead of the real names of the KB artifacts, which will mitigate the OOV problem. In consequence, the model could form the actions more precisely.

To further decrease search space, the copy mechanism is also incorporated. The copy mechanism *replicates* all masked symbols in the input sequence to form the output, instead of letting the decoder generate them from the decoder vocabulary. As a result, the decoder only needs to generate primitive actions, further reducing vocabulary size to $O(17^N)$.

The benefits of our design are twofold. (1) The much-reduced vocabulary makes convergence faster, as the generator is only concerned about generating correct primitive actions, but not names of artifacts from the KB. (2) Solve questions with unforeseen KB artifacts by directly masking and copying them from the input question when generating an action sequence.

Encoder. The encoder is a bidirectional LSTM that takes a question of variable length as input and generates an encoder vector $\boldsymbol{e}_i$ at each time step i.

$$\boldsymbol{e}_i, \boldsymbol{h}_i = LSTM(\phi_E(x_i), \boldsymbol{h}_{i-1}). \tag{1}$$

Here ϕ_E is word embedding of token E. $(\boldsymbol{e}_i, \boldsymbol{h}_i)$ is the output and hidden vector of the i-th time step when encoding. The dimension of $\boldsymbol{e}_i$ and $\boldsymbol{h}_i$ are set as the same in this work. $\boldsymbol{e}_i$ is the concatenation of the forward $(\boldsymbol{e}_i^F)$ and backward $(\boldsymbol{e}_i^B)$ output vector and $\boldsymbol{h}_i$ is the encoder hidden vector. The output vectors $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T)$ is regarded as a short-term memory $\boldsymbol{M}$, which is saved to use in copy mode.

Decoder. Our decoder of NS-CQA predicts output tokens following a mixed probability of two models, namely **generate-mode** and **copy-mode**, where the former generates tokens from the fixed output vocabulary and the latter copies words from the input tokens. Furthermore, when updating the hidden state

at time step t, in addition to the word embedding of predicted word at time $t-1$, the location-based attention information is also utilised.

By incorporating the copy mechanism, the generated actions might be chosen from the output vocabulary or input tokens. We assume a fixed output vocabulary $\mathcal{V}_{output} = \{v_1, \ldots, v_N\}$, where $\mathcal{V}_{output}$ contains operators and related arguments in action sequence. In addition to that, all the unique words from input tokens $x = \{x_1, \ldots, x_T\}$ constitute another set $\mathcal{X}$ whereby some OOV words could be 'copied' when such words are contained in $\mathcal{X}$ but not in $\mathcal{V}_{output}$. Therefore, the vocabulary unique to input tokens x is: $\mathcal{V}_x = \mathcal{V}_{output} \cup \mathcal{X}$.

The **generate-mode** predicts output token a_t from the output vocabulary $\mathcal{V}_{output}$. Like traditional Seq2Seq model, the decoder is another LSTM model that generates a hidden vector $\boldsymbol{q}_t$ from the previous output token a_{t-1}. Previous step's hidden vector $\boldsymbol{q}_{t-1}$ is fed to an attention layer to obtain a context vector $\boldsymbol{c}_t$ as a weighted sum of the encoded states. Current step's $\boldsymbol{q}_t$ is generated via:

$$\boldsymbol{q}_t = LSTM(\boldsymbol{q}_{t-1}, [\phi_D(a_{t-1}), \boldsymbol{c}_t]) \tag{2}$$

Here ϕ_D is the word embedding of input token a_{t-1}. The dimension of $\boldsymbol{q}_t$ is set as d_q, and the attention weight matrix is trainable. The hidden vector $\boldsymbol{q}_t$ is used to compute the score of target word v_i in $\mathcal{V}_{output}$ as:

$$\psi_g(a_t = v_i) = \boldsymbol{v}_i^\top \boldsymbol{W}_o \boldsymbol{q}_t, \; v_i \in \mathcal{V}_{output} \tag{3}$$

where $\boldsymbol{W}_o$ is trainable matrix and $\boldsymbol{v}_i$ is the vector of word v_i.

In **copy-mode**, the score of "copying" word x_j from input tokens $\{x_1, \ldots, x_T\}$ is computed as:

$$\psi_c(a_t = x_j) = \sigma(\boldsymbol{e}_j \boldsymbol{W}_c \boldsymbol{q}_t), \; x_j \in \{x_1, \ldots, x_T\} \tag{4}$$

where $\boldsymbol{W}_c \in \mathbb{R}^{d_q \times d_q}$, σ is a non-linear activation function and, hidden encoder vectors $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}$ in short-term memory $\boldsymbol{M}$ are used to map the input tokens $\{x_1, \ldots, x_T\}$ respectively.

Finally, given the hidden vector $\boldsymbol{q}_t$ at time t and short-term memory $\boldsymbol{M}$, the output token a_t is generated following a mixed probability as follows:

$$p(a_t | \boldsymbol{q}_t, a_{t-1}, \boldsymbol{M}) = p_{cg}(a_t | \boldsymbol{q}_t, a_{t-1}, \boldsymbol{M}) \tag{5}$$

where p_g and p_c indicate the generate-mode and copy-mode respectively. They are calculated as follows:

$$p_{cg} = \begin{cases} \frac{1}{Z} e^{\psi_g(a_t)}, & a_t \in \mathcal{V}_{output} \\ \frac{1}{Z} \Sigma_{j: x_j = a_t} e^{\psi_c(a_t)}, & a_t \in \mathcal{X} \end{cases} \tag{6}$$

where Z is the normalization term and is computed as: $Z = \Sigma_{v \in \mathcal{V}_{output}} e^{\psi_g(v)} + \Sigma_{x \in \mathcal{X}} e^{\psi_c(x)}$.

At time step t, word embedding of previous output token a_{t-1} and the location-based attention information are both employed to update the hidden state. a_{t-1} will be represented as $[\phi_D(a_{t-1}); \boldsymbol{r}_{q_{t-1}}]$, where $\phi_D(a_{t-1})$ is the word embedding of a_{t-1} and $\boldsymbol{r}_{q_{t-1}}$ is the weighted sum of hidden states $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}$ in $\boldsymbol{M}$.

Vector $\boldsymbol{r}_{q_{t-1}}$ is calculated as:

$$\boldsymbol{r}_{q_{t-1}} = \Sigma_{\tau=1}^{T} \rho_{t\tau} \boldsymbol{e}_\tau \tag{7}$$

$$\rho_{t\tau} = \begin{cases} \frac{1}{K} p_{cg}(x_\tau | \boldsymbol{q}_{t-1}, a_{t-1}, \boldsymbol{M}), & x_\tau = a_{t-1} \\ 0, & otherwise \end{cases} \tag{8}$$

where K is the normalization term which is $\Sigma_{\tau': x_{\tau'} = a_{t-1}} p_{cg}(x_{\tau'} | \boldsymbol{q}_{t-1}, a_{t-1}, \boldsymbol{M})$, considering there might be input tokens located at different positions which equal to a_{t-1}. $\rho_{t\tau}$ is viewed as location-based attention in our work.

3.4. Symbolic Executor

A symbolic executor is implemented as a collection of deterministic, generic functional modules to execute the primitive actions, which have a one-to-one correspondence with the functional modules. The symbolic executor first analyzes the output tokens produced by neural generator, and would assemble the actions one by one. Given an action sequence that begins with the first action, the symbolic executor executes the actions in order, on the intermediate result of the previous one. As discussed in Section 3.1, this is only possible due to our carefully designed primitive actions. Otherwise, complex memory mechanisms would need to be incorporated to maintain intermediate answers [28, 8]. Upon encountering the action *EOQ*, the result from the last execution step will be returned as the final answer.

3.5. Training Paradigm

As the symbolic executor executes non-differentiable operations against a KB, it is difficult to utilize end-to-end back-propagation to optimize the neural generator. Therefore, we adopt the following two-step procedure to train the generator. By using a breadth-first-search (BFS) algorithm, we generate pseudo-gold action sequences for a tiny subset of questions. In BFS, we assemble all the operators and KG artifacts found in question to form candidate action sequences in a brute-force way. We then execute the candidate action sequences to find the ones that yield the right answer and view them as pseudo-gold action sequences. Using these pairs of questions and action sequences, we pre-train the model by Teacher Forcing.

We then employ RL to fine-tune the generator on another set of question-answer pairs. The symbolic executor executes the predicted action sequence to output an answer and yield a reward for RL. The reward is the similarity between the output answer and the gold answer.

As shown in Algorithm 1, our method works as follows. The training starts with an empty memory buffer, and at every epoch, for each sample, the generated trials that gain high reward will be stored in the memory. For each question, we first use a search algorithm, i.e., greedy-decoding, to generate a trial. We execute the trial and compute a reward r_{greedy}, which is set as the reward threshold. Then we employ a beam search method to generate a set of candidate trials for the question and compute their rewards. At every epoch, we compare the generated trials with the trials in memory to determine the reward bonus, aka proximity and novelty, and further add the reward bonus to the adaptive reward. A candidate trial is added into the memory buffer if its reward is higher than r_{greedy}. We utilize the augmented reward to train the policy under the RL setting.

The memory buffer stores a limited set of trials for the training questions. Once the memory buffer is full, we substitute a random trial with the current new trial. This strategy enables the memory buffer to maintain relatively fresher trials, but would not always abandon the older ones.

Algorithm 1: Training NS-CQA

Input: Training dataset Q_{train}, initial policy θ, memory buffer M, reward function $R(\cdot)$, learning rate η_1
Output: The learned policy θ^*

Randomly initialize θ
$M \leftarrow \emptyset$
while *not converged* **do**
  Sample batch of data $Q_{batch} \sim Q_{train}$
  $\mathcal{L} \leftarrow 0$
  for $q \in Q_{batch}$ **do**
    Get one trial $\boldsymbol{t_{greedy}}$ by greedy-decoding
    Compute adaptive reward r_{greedy}
    Compute cumulative reward $R(q, \boldsymbol{t_{greedy}})$
    Sample K trials: $\boldsymbol{t_k} \sim \pi(\boldsymbol{t}|q;\theta)$
    for *each trial* $\boldsymbol{t_k}$ **do**
      Compute adaptive reward $r_{\boldsymbol{t_k}}$
      Compute cumulative reward $R(q, \boldsymbol{t_k})$
      Update memory: add $\boldsymbol{t_k}$ to M if $r_{\boldsymbol{t_k}} > r_{greedy}$
    end
    $L = \frac{1}{K}\sum_{k=1}^{K}[R(q, \boldsymbol{t_k}) - R(q, \boldsymbol{t_{greedy}})]log(p_\theta(\boldsymbol{t_k}))$
    $\mathcal{L} \leftarrow \mathcal{L} + L$
  end
  Compute adapted parameters: $\theta \leftarrow \theta + \eta_1 \nabla_\theta \mathcal{L}$
end
Return The learned policy $\theta^* \leftarrow \theta$

The main components of our training paradigm, namely the RL method, the adaptive reward, and the curriculum reward bonus, are described in the rest of this section.

Reinforcement Learning. In this step, REINFORCE [41] is used to finetune the neural generator. Typically, the three notions mentioned in RL are action, state, and reward, respectively. In our scenario, at each step, action as a fundamental concept in RL is a token produced by a neural generator used to form an executable action sequence. What needs to be clarified is that when related to RL, the concept of actions refers to the tokens of a trial. Since the complex question answering environment is deterministic, we define the state as the question combined with the generated tokens so far. Meanwhile, the reward is the same as the notion in RL.

In our work, the state, action and reward at time step t are denoted as s_t, a_t and r_t respectively. Given a question q, the state of time step t is defined by q and the action sequence so far: $s_t = (q, a_{0:t-1})$, and the action tokens are generated by the generator. At the last step of decoding T, the entire sequence of actions is generated. The symbolic executor will then execute the action sequence to produce an output answer ans_o. Therefore the reward is computed only after the last step of decoding when ans_o is output. We design a Adaptive Reward Function (ARF), which is the adaptive comparison of the output answer ans_o and the gold answer ans_g. Furthermore, we employ a curriculum-guided Reward Bonus (CRB), which comprises proximity and novelty, to assign non-zero rewards for actions that do not yield correct answers. Specifically, the cumulative reward of an action sequence $a_{0:T}$ is the sum of CRB and ARF:

$$R(q, a_{0:T}) = CRB + ARF(ans_o, ans_g) \tag{9}$$

Then $R(q, a_{0:T})$ is sent back to update parameters of the neural generator through a REINFORCE objective as the supervision signal.

Adaptive Reward. Though the search space is significantly reduced to $O(17^N)$ after the masking and copy mechanisms are incorporated, the length of action sequence, which is N, would be fairly long when solving a relatively more complex question. In that case, $O(17^N)$, the size of the search space, is still huge which makes it hard for the model to find correct action sequences when gold annotations are unavailable.

Moreover, since the reward used to train the model could only be obtained after a sequence of actions is executed, the execution of actions is regarded as a part of training. Suppose a large amount of candidate action sequences (normally more than 50) are generated, their execution would consume a large amount of time since it involves searching triples in KB, performing set operations and discrete reasoning. To reduce the training time, we limit our NS-CQA model to form only 5∼20 candidate action sequences with a smaller beam size. With the huge search space and small beam size, the sparsity of the reward becomes a problem. Of all the candidate action sequences that are predicted, very few of them could output correct answers and be positively rewarded while most of them do not produce any reward. Under such circumstances, without the notion of partial reward, the neural generator would suffer from high variance and instability. Therefore the generator would be inclined to be trapped in local optima and not generalize well on data never seen before.

Furthermore, different categories of questions entail different answer types. The reward function should be adaptive to the diverse answer types which could measure the degree of correctness of predicted answers more precisely. In other words, the reward function should encourage the generator to generate action sequences with the correct answer type while punishing the model if the predicted answer type is incorrect.

In the complex question answering scenario, the possible types of answers are integers, sets of entities and Boolean values. To measure the answer correctness more accurately and adaptively, and to compute partial reward to alleviate the sparse reward problem, we define our adaptive reward function ARF. ARF computes reward based on different answer types using the function Sim between the output answer ans_o and the gold answer ans_g.

$$Sim(ans_o, ans_g) = \begin{cases} 1 - \frac{|ans_g - ans_o|}{|ans_g + ans_o + \varepsilon|}, & \text{integer} \\ Edit(ans_g, ans_o), & \text{Boolean} \\ F1(ans_g, ans_o), & \text{set} \end{cases} \tag{10}$$

The edit-score is used to measure the accuracy of the output provided the answer type is Boolean, while the F1-score is used as a reward when answer is a set of entities. When the answer type is Boolean, the expected output is a list of Boolean value, for instance as what is presented in Table 6, the expected answer

of the question "s Alda Pereira-Lemaitre a citizen of France and Emmelsbull-Horsbull?" is [True, False]. Regard each Boolean value as a single element in a list, the edit (Levenshtein) distance is used to compute the similarity between two lists, i.e., output answer list, and gold answer list. Thus the similarity is calculated as follows:

$$Edit(ans_g, ans_o) = 1 - \frac{Levenshtein(ans_g, ans_o)}{max(|ans_g|, |ans_o|)} \quad (11)$$

On the other hand, suppose the type of answer is a set of entities, F1-score is computed as:

$$F1(ans_g, ans_o) = 2*\frac{precision * recall}{precision + recall} = 2*\frac{\frac{|ans_g \cap ans_o|}{|ans_o|} * \frac{|ans_g \cap ans_o|}{|ans_g|}}{\frac{|ans_g \cap ans_o|}{|ans_o|} + \frac{|ans_g \cap ans_o|}{|ans_g|}} \quad (12)$$

If the answer type is incorrect or the action sequence is sematically invalid, reward is set as 0. On the other hand, if the answer type is the same as the gold answer, partial reward is granted. We then defined ARF as follows:

$$ARF(ans_o, ans_g) = R_{type} * (W_1 + W_2 * Sim(ans_o, ans_g)) \quad (13)$$

In our work, ε, W_1 and W_2 are predefined hyper-parameters and set as 0.001, 0.2 and 0.8 repectively. If the type of predicted answer is correct, R_{type} is set as 1, otherwise 0.

Curriculum-guided Reward Bonus. In many RL settings, the reward is positive only when a trial, i.e., a long sequence of actions generated by a policy, could yield the correct result. At the initial stage, since the policy is not yet fully-trained, out of all the generated trials, the rate of successful trials is rare. Thus, there is an insufficient number of collected successful trials for training the RL model, which causes the sparse reward problem.

Besides, the suboptimal policy will not explore the search space effectively since many sampled trials could be repeated. Moreover, the policy will forget the rare successful trials easily since the trials may not be re-sampled frequently. These factors often lead to the data inefficiency problem.

To solve the above problems, we design two reward bonuses to learn from failed trials. We introduce a memory buffer to record the high-reward trials for each training sample. We compare a generated trial with those stored in the memory buffer to see how similar it is to the recorded trials. Therefore we give *proximity bonus* to a generated trial even if it fails to yield the correct answer. By doing this, we could encourage the policy to re-sample the high-reward trials and accordingly reduce the frequency of generating infeasible trials. Also, we give *novelty bonus* to the generated trials that differ from the trials in the memory buffer. The novelty bonus encourages the policy to generate different trials, thus avoid being trapped by spurious ones. Once the reward is augment with the above two bonuses, the corresponding failed experience is assigned with a nonnegative reward and can contribute to learning the policy.

To balance proximity and novelty, we employ a curriculum-learning method to regulate their trade-off dynamically. In the earlier epochs, higher novelty can help the policy to explore unseen areas and generate more diverse trials. However, in the later epochs, such novelty will bring more noise (often as spurious trials) into training and distract the policy. At the later stage of training, the policy has gained sufficient knowledge about the tasks and is able to generate high-reward, promising trials. Therefore, proximity becomes more critical since it will encourage the policy to proceed towards the correct trials and to focus on learning how to generate promising trials.

Given a question q, suppose the high-reward trials $t_1^q, \ldots, t_m^q$ have already been stored in the memory buffer M. For one generated trial t, we compute the reward bonus CRB as:

$$CRB = \alpha(\lambda F_{prox}(t, M) + (1 - \lambda)F_{novel}(t, M)), \quad (14)$$

where $\alpha \in [0, 1]$ is the weight of the reward bonus and dependent on the scale of the task rewards. The term F_{prox} reflects the proximity of the trial t to the recorded trials in memory M while F_{novel} measures the novelty of t. The value of λ controls their relative proportion, which is adjusted by the curriculum learning method.

We compute the similarity between the generated trial t with one trial t_i^q in M by edit distance, which is:

$$s_i = Edit(t, t_i^q) \quad (15)$$

Thus we define the proximity F_{prox} as the highest similarity between the trial t and all the trials $t_1^q, \ldots, t_m^q$ in the memory buffer, which is:

$$F_{prox}(t, M) = \max_{1 \leqslant i \leqslant m} (s_i) \quad (16)$$

The term novelty F_{novel} measures the diversity of the trial t from the trials $t_1^q, \ldots, t_m^q$. We assign a high novelty to a generated trial if it is different from the trials in M, thus we define the novelty as:

$$F_{novel}(t, M) = \beta - \frac{1}{m}\Sigma_{i=1}^{m} s_i, \quad (17)$$

where $\beta \in [0, 1]$ is used to measure the diversity and is dependent on the scale of the similarity.

We employ a curriculum learning scheme to adaptively change the weight λ in Formula 14. We start from learning to generate novel trials with large diversity, and gradually focus on re-sampling the trials which have high proximity to the desired successful trials stored in the memory buffer. This method is achieved by progressively increasing the weight λ, which is exponentially increased λ with the training epochs:

$$\lambda = min\{1, (1 + \eta)^{\gamma}\lambda_0\}, \quad (18)$$

where $\eta \in [0, 1]$ is the learning pace which controls the curriculum learning, γ represents the number of the epochs that have been trained, and λ_0 is the initial weight of λ.

In our work, α, β, η, and λ_0 are hyper-parameters which are defined as 0.1, 1.0, 0.08, and 0.1, respectively in our work.

REINFORCE. At each time step, the output token generated by agent is decided by a certain policy (which is the generator

in our work), and the probability that one token a is chosen is computed as below, where θ denotes model parameters:

$$\pi_\theta(q, a) = P_\theta(a_t = a|q, a_{0:t-1}) \tag{19}$$

Thus, the probability of an entire action sequence $a_{0:T}$ is given by:

$$P_\theta(a_{0:T}|q) = \prod_{t=1}^{T} P_\theta(a_t|q, a_{0:t-1}) \tag{20}$$

In 9, we define the cumulative reward $R(q, a_{0:T})$. Our objective is to maximize the expected cumulative reward. Therefore we use the policy gradient method such as the REINFORCE algorithm to finetune the generator. The objective and gradient are:

$$\begin{aligned} J^{RL}(\theta) &= \sum_q \mathbb{E}_{P_\theta(a_{0:T}|q)}[R(q, a_{0:T})] \\ \nabla_\theta J^{RL}(\theta) &= \sum_q \sum_{a_{0:T}} P_\theta(a_{0:T}|q) \cdot [R(q, a_{0:T}) \\ &\quad - B(q)] \cdot \nabla_\theta log P_\theta(a_{0:T}|q) \end{aligned} \tag{21}$$

$B(q) = R(q, \hat{a}_{0:T})$ is a baseline that reduces the variance of the gradient estimation without introducing bias. In our work, the baseline is set as what is used in the self-critical sequence training (SCST) [42]. Also, Monte Carlo integration is employed to approximate the expectation over all possible trials in the policy gradient method [41]. The training method is presented in Algorithm 1.

4. Experiments

We evaluated our model NS-CQA on a large-scale complex question answering dataset (CQA) [6], and a challenging multi-hop question answering dataset WebQuestionsSP [17].

The CQA dataset is generated from the facts stored in Wikidata [12], consisting of 944K QA pairs for training and 100K/156K QA pairs for validation and test, respectively. The CQA dataset is characterised by the challenging nature of the questions. To answer them, discrete aggregation operators such as set union, intersection, min, max, counting, etc. are required (see Table 2 for more details). The CQA questions are organized into seven categories, as shown in Table 3. Some of these categories (e.g., Simple Question) have entities as answers, while others have numbers (e.g., Quantitative (Count)) or Boolean values (e.g., Verification (Boolean)) as answers. We used 'accuracy' as the evaluation metric for categories whose answer type is 'Verification', 'Quantitative (Count)', and 'Comparative (Count)'; and 'F1 measure' for other types of questions. However, to simplify the presentation and stay consistent with literature [7, 9], we denote 'accuracy' as 'F1 measure' in Table 3. Hence, the model performance was evaluated on the F1 measure in this paper. Furthermore, we computed the micro F1 and macro F1 scores for all the models based on the F1-scores of the seven question categories.

In our analysis of the CQA dataset, we found that the seven categories of questions vary substantially in complexity. We found that 'Simple' is the simplest that only requires two actions to answer a question, whereas 'Logical Reasoning' is more difficult that requires three actions. Categories 'Verification', 'Quantitative Reasoning', and 'Comparative Reasoning' are the next in the order of difficulty, which need 3–4 actions to answer. The most difficult categories are 'Quantitative (Count)' and 'Comparative (Count)', needing 4–5 actions to yield an answer. Saha et al. [7] drew a similar conclusion through manual inspection of these seven question categories.

The WebQuestionsSP dataset collects multi-hop questions, i.e., the questions require a chain of KB triples to answer, via the Google Suggest API. In comparison to the CQA dataset, WebQuestionsSP can be considered easier as it only contains multi-hop questions, and the answers are (sets of) entities only. It consists of 3,098 question-answer pairs for training and 1,639 questions for testing. We utilized the same evaluation metrics employed by [6, 8], the F-1 measure, to evaluate model performance on the testing questions.

4.1. Model Description

Our model is evaluated against three baseline models: HRED+KVmem [6], NSM [8] and CIPITR [7]. We used the open-source code of HRED+KVmem and CIPITR to train the models and present the best result we obtained. As the code of NSM has not been made available, we re-implemented it and further incorporated the copy and masking techniques we proposed. HRED+KVmem does not use beam search, while CIPITR, NSM, and our model all do for predicting action sequences. When inferring the testing samples, we used the top beam [7], i.e., the predicted program with the highest probability in the beam, to yield the answer.

HRED+KVmem [6] is the baseline model proposed together with the CQA dataset [6], which combines a hierarchical encoder-decoder with a key-value memory network. The model first encodes the current sentence with context into a vector, whereby a memory network retrieves the most related memory. Then the retrieved memory is decoded to predict an answer from candidate words. HRED+KVmem was designed specifically for the CQA dataset, thus was not included in our experiments on WebQuestionsSP.

NSM [8] is an encoder-decoder based model which is trained by weak-supervision, i.e., the answers to the questions. NSM first employs an Expectation-Maximization-like (EM-like) method to find pseudo-gold programs that attain the best reward. It iteratively uses the current policy to find the best programs and then maximizes the probability of generating such programs to optimize the policy. Then NSM replays *one* pseudo-gold trial that yields the highest reward for each training sample when employing REINFORCE to train the policy. It assigns a deterministic probability to the best trial found so far to improve the training data efficiency. NSM was at first proposed to solve the problems in WebQuestionsSP, and we reimplemented it to also handle the CQA dataset.

As presented in 3.1, unlike NSM, we do not refer to the intermediate variables when generating the tokens of a trial. Therefore it is unnecessary to incorporate the key-variable

Table 3: Performance comparison (measured in F1) of the four methods on the CQA test set. Best results for each category is **bolded**, and second best is underlined.

Method	HRED+KVmem	CIPITR-All	CIPITR-Sep	NSM	Vanilla	PG	NS-CQA
Simple Question	41.40%	41.62%	**94.89%**	88.33%	85.13%	84.25%	88.83%
Logical Reasoning	37.56%	21.31%	**85.33%**	81.20%	70.46%	68.37%	81.23%
Quantitative Reasoning	0.89%	5.65%	33.27%	41.89%	47.96%	56.06%	**56.28%**
Comparative Reasoning	1.63%	1.67%	9.60%	64.06%	54.92%	**67.79%**	65.87%
Verification (Boolean)	27.28%	30.86%	61.39%	60.38%	75.53%	83.87%	**84.66%**
Quantitative (Count)	17.80%	37.23%	48.40%	61.84%	66.81%	75.69%	**76.96%**
Comparative (Count)	9.60%	0.36%	0.99%	39.00%	34.25%	43.00%	**43.25%**
Overall macro F1	19.45%	19.82%	47.70%	62.39%	62.15%	68.43%	**71.01%**
Overall micro F1	31.18%	31.52%	73.31%	76.01%	74.14%	76.56%	**80.80%**

memory, which is used to maintain and refer to intermediate program variables in our work. We thus removed the key-variable memory component in the seq2seq model in our reimplementation of NSM.

CIPITR [7] employs an NPI technique that does not require gold annotations. Instead, it relies on auxiliary awards, KB schema, and inferred answer types to train an NPI model. CIPITR transforms complex questions into neural programs and outputs the answer by executing them. It designs high-level constraints to guide the programmer to produce semantically plausible programs for a question. The auxiliary reward is designed to mitigate the extreme reward sparsity and further used to train the CIPITR model. CIPITR is designed to handle the KBQA problems proposed in both CQA and WebQuestionsSP.

4.2. Training

The NS-CQA model was implemented in PyTorch with the model parameters randomly initialized[1]. The Adam optimizer is applied to update gradients defined in Formula 21. We used the fixed GloVe [39] word vectors to represent each token in input sequences and set each unique, unseen word a same fixed random vector. We set a learning rate of 0.001, a mini-batch size of 32 samples to pre-train the Seq2Seq model with pseudo-gold annotations. On average, after about 70 epochs, the Seq2Seq model would converge. Then we trained the REINFORCE model with a learning rate of 1e-4 and a mini-batch size of 8 on the pre-trained Seq2Seq model until accuracy on the validation set converged (at around 30 epochs).

As solving the entity linking problem is beyond the scope of this work, we separately trained an entity/class/relation linker. When training the NS-CQA model, the predicted entity/class/relation annotations along with the pseudo-gold action sequence (which are generated by a BFS algorithm) were used. The entity/class/relation annotations predicted by the respective linker were used when conducting experiments on the test dataset.

[1]To encourage reproductivity, we have released the source code at `https://github.com/DevinJake/NS-CQA`.

Incorporating the copy and masking mechanisms, our full model took a total of at most 3,700 minutes to train 100 epochs (70 epochs for the Seq2Seq model and 30 epochs for REINFORCE) till convergence. Most of the time was spent on RL training, which is over 3,633 minutes. In constraints, when we tried to train CIPITR [7], the model required over 24 hours to complete one epoch of training while the max number of epochs is also set as 30.

Training with annotations would make the model learn to search in a relatively more accurate space, thus converging faster. However, the limited availability of annotations remains a bottleneck for model training in many CQA tasks. On the other hand, training without annotations but with denotations solely makes model convergence harder.

We married the two ideas together: training with a small number of annotations and then the denotations. First, we automatically produced pseudo-gold annotations for a small set (e.g., less than 1% of the entire CQA training dataset) of questions. The pseudo-gold annotations were utilized to pre-train the model to constrain the search space. After that, the model was further trained with only denotations.

4.3. Results on CQA

Table 3 summarizes the performance in F1 of the four models on the full test set of CQA.

It must be pointed out that **CIPITR** [7] separately trained *one single model for each of the seven question categories*. We denote the model learned in this way as **CIPITR-Sep**. In testing, CIPITR-Sep obtained test results of each category by employing the corresponding tuned model [7]. In practical use, when trying to solve a complex question more precisely, CIPITR-Sep has to first trigger a classifier to recognize the question category. Only after acquiring the question categories information could CIPITR-Sep know which model to select to answer the question. If the number of question categories is increased, CIPITR-Sep needs to train more models, which will impede the system from generalizing to unseen instances. Besides, CIPITR also trained *one single model over all categories of training examples* and used this single model to answer all questions. We denote this single model as **CIPITR-All**. Therefore, we separately present the performance of these two variations of CIPITR in Table 3. On the other hand, we tuned NS-CQA on all

categories of questions with one set of model parameters. Our model is designed to adapt appropriately to various categories of questions with one model, thus only needs to be trained once.

We also compared our full model, NS-CQA, with several model variants to understand the effect of our techniques presented in this work. Specifically, **Vanilla** is an imitation-learning model that was trained with pseudo-gold annotations. **PG** denotes the RL model that was optimized by the Policy Gradient algorithm based on the pre-trained model Vanilla. **NS-CQA** means the RL model that is equipped with all the techniques proposed in this work, notably the memory buffer and the reward bonus.

In Table 3, several important observations can be made.

1. Over the entire test set, our full model NS-CQA achieves the best overall performance of 71.01% and 80.80% for macro and micro F1, respectively, outperforming all the baseline models. The performance advantage on macro F1 over the four baselines is especially pronounced, by 51.56, 51.19, 23.31, and 8.62 percentage points over HRED+KVmem, CIPITR-All, CIPITR-Sep, and NSM respectively. Also, NS-CQA improves over the micro F1 performance of HRED+KVmem, CIPITR-All, CIPITR-Sep, and NSM by 49.62%, 49.28%, 7.49%, and 4.79%. Moreover, our model achieves best or second-best in all the nine items being evaluated (the seven categories, plus overall macro F1, and overall micro F1). The improvement is mainly due to the techniques presented in this work. We introduce masking and copy mechanisms to reduce the search space effectively and carefully design a set of primitive actions to simplify the trials, therefore enable the model to efficiently find the optimal trials. We also augment the RL model with a memory buffer, whereby the model could circumvent the spurious challenge, and remember the high-reward trials to re-sample them.
2. Out of the seven categories, our full model NS-CQA achieves the best performance in four categories: Quantitative Reasoning, Verification (Boolean), Quantitative (Count), and Comparative (Count), and second best in the rest three. In the hardest categories, Quantitative (Count) and Comparative (Count), NS-CQA is substantially superior over the four baseline models, and outperforms our PG model. Since the length of the questions in the hardest categories is usually higher than in the other categories, it is always hard to find correct trials. Under such circumstances, the memory buffer could make the model search in unknown space while keeping the previous high-reward trials in mind, which makes the model easier to train. This is the main reason that NS-CQA performs the best in the hardest categories.
3. CIPITR-Sep achieves the best performance in two *easy* categories, including the largest type, Simple Question. For the harder categories, it performs poorly compared to our model. Also, CIPITR-All, the single model that is trained over all categories of questions, performs much worse in all the categories than CIPITR-Sep, which learns a different model separately for each question category. For CIPITR-Sep, the results reported for each category are obtained from the model explicitly tuned for that category. A possible reason for CIPITR-All's significant performance degradation is that the model tends to forget the previously appeared high-reward trials when many infeasible trials are generated. Besides, the imbalanced classes of questions also deteriorates the performance of the model. Different from CIPITR, our model is designed to remember the high-reward trials when training.
4. NSM and NS-CQA both produce competitive results. The copy mechanism, masking method, and our carefully-defined primitive actions presented in this work were used in both models when we implemented them. By comparing the overall macro and micro F-1 score, it could be observed that NSM performed the best in all the four baseline models. This helps to validate the effectiveness of our proposed techniques. However, NSM is worse than NS-CQA in all categories, especially in the harder ones. Since NSM records one promising trial for each question, it might be faced with the spurious problem. Different from NSM, we design a memory buffer for recording all successful trials to circumvent this problem. Also, NSM only considers the correctness of the predicted answers when measuring the reward, hence suffers from the sparse reward problem. Unlike NSM, our NS-CQA model augments the reward with proximity and novelty to mitigate this problem. These two factors make our model superior to the NSM model in all question categories.
5. Both of our model variants perform competitively. In Table 3, it can be seen that the PG model, which was equipped with RL, performed better than the Vanilla model in five categories, but did not perform well in the two easy categories, Simple and Logical Reasoning. We analyzed the degeneration and found that for these two types of questions, usually, each question has only one correct sequence of actions. When training with PG, some noisy spurious trials were introduced by beam search and thus degraded the model's performance. Our full model is better than the PG model in six of the seven categories and substantially improved performance in the Logical Reasoning category. We compared the trials generated by the full model and the PG model, and found that many noisy trials are removed with the help of the memory buffer. That is the main reason for the improvement in the Logical Reasoning category. However, we also found that the full model performed worse than PG in Comparative Reasoning, which will be further investigated in the future.

The above results demonstrate the effectiveness of our technique. It is worth noting that our model is trained on only 1% of the training samples, whereas the baseline models use the entire training set. Besides, our approach uses one model to solve all questions, while CIPITR-Sep trains seven separate models to solve the seven categories of questions. Thus our model is virtually compared with seven individually training models used in CIPITR-Sep. However, our model still achieves best performance overall as well as in five of the seven categories.

Table 4: Performance comparison (measured in F1) of the four methods on the WebQuestionsSP test set. Best results is **bolded**.

Method	F1 measure
CIPITR-All	43.88%
NSM	70.61%
PG	70.72%
NS-CQA	**72.04%**

4.4. Results on WebQuestionsSP

Table 4 summarizes the performance in the F1 measure of the four models on the full test set of WebQuestionsSP.

Similar to the CQA dataset, CIPITR also divided questions into five categories, and then separately train one model for each category. However, since the category information is not provided in the WebQuestionsSP dataset, we did not classify the questions and trained one single model, CIPITR-All, for all the training samples by using its open-source code.

From Table 4, we can observe that NS-CQA can indeed learn the rules behind the multi-hop inference process directly from the distance supervision provided by the question-answer pairs. Without manually pre-defined constraints, our model could learn basic rules from the pseudo-gold annotations, and further complete the rules by employing RL.

NS-CQA performed the best in the four models and significantly outperformed the CIPITR-All. The main reason is that it is hard for CIPITR-All to learn one set of parameters that fits the different samples.

Also, by introducing the masking and copy mechanism, NSM could achieve a performance competitive to our models PG and NS-CQA. By employing memory buffer, our NS-CQA model can alleviate the sparse reward problem and avert being trapped by spurious trials, which makes the model more robust, therefore achieving the best performance.

Furthermore, NS-CQA achieves the best result on both the CQA and WebQuestionsSP datasets, which attests to the effectiveness and the generalizability of our method.

4.5. Model Analysis

To study how the different components influence the performance of our seq2seq model, i.e., Vanilla, we conduct an ablation experiment as follows. Each of the main components: attention, copy mechanism, and masking method, is removed one at a time from the full seq2seq model to study how its removal affects model performance. We also study the effect of smaller training samples on performance, by using 1K and 2K samples for training, instead of 10K used in the full model.

Table 5 summarizes performance degradation on the CQA test set, where the Vanilla model achieves a macro F1 score of 62.15%. It can be seen that the removal of masking produces the largest drop in performance, of 37.10%. Masking method significantly decreases the search space by replacing all the entity, relation and type names with wildcard tokens. This result demonstrates that although a simple approach, masking proves to be valuable for the CQA task.

Table 5: Ablation study on the CQA test set, showing the macro F1 score drop by removing each main component, or by learning from a subset of the training set. The Vanilla model has macro F1 of 62.15% as shown in Table 3.

Feature	Macro F1
Vanilla	62.15%
Masking	-37.10%
Copy	-11.52%
Attention	-4.30%
1,000-training	-5.78%
2,000-training	-4.09%

When the copy mechanism is removed, performance decreases by 11.52%. This is consistent with our expectation since masking has already considerably decreased the search space, the improvements that copy mechanism could makes is relatively limited. Lastly, when training on fewer samples labeled with pseudo-gold actions, the model under-fits.

When training on even smaller datasets, the performance degradation is not as severe as we expected. With as a training set as small as 1,000, our model is able to generalize well, only suffering a 5.78% drop on a test set of 15.6K. With a training set of 2,000 samples, our model suffers a modest 4.09% drop in performance. This study further demonstrates the robustness and generalizability of our model.

4.6. Sample Size Analysis

In this subsection, we analyze the effect of training samples of different sizes on our PG module. Since the WebQuestionsSP consists of a limited number of questions, it is hard to conduct the sample size analysis on it. Instead, we trained our model by using different CQA subsets to make a comparison. Specifically, given the same pre-trained model, we train the NS-CQA model on 0.2%, 0.4%, 0.6%, 0.8%, 1.0%, and 1.2% of total 944K training samples. Note that the evaluation results of the full model presented in Section 4.3 are obtained from 1.0% of training data and the entire test set (i.e., 156K). For experiments described in this subsection, evaluation is performed on a subset of the full test set that is 10% of its size (i.e., 15.6K). Training of the REINFORCE model is stopped at 30 epochs, which is when all models have been observed to converge.

We first study the effect on model performance. Figure 3 plots the macro F1 values of the seven categories of questions as well as the overall performance. With the increase in training data size, a general upward trend in macro F1 values can be observed, with the category Simple Question being the exception. For the overall test set, we can observe that the macro F1 value plateaus at 1.0% and does not increase when training data is expanded to 1.2%.

For Simple Question, the output actions are relatively the 'simplest'. In most cases, one 'Select' action is needed to solve a question. Consequently, with the help of methods to decrease search space and better use data, after pre-training, the model overfits on the Simple Question type rapidly. Therefore the performance of answering Simple Question fluctuates with the change of training sample size.

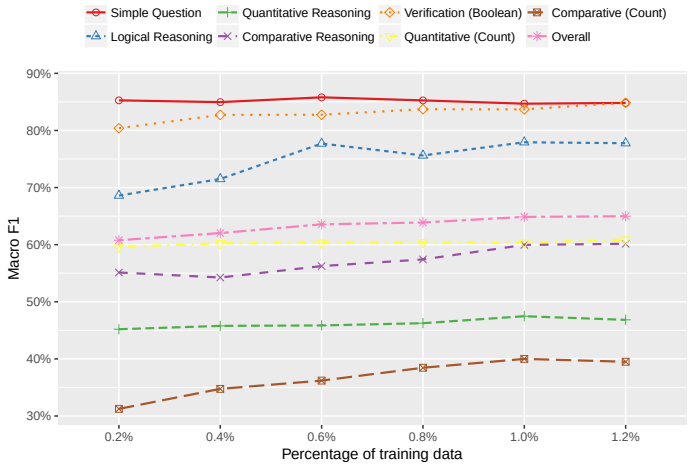

Figure 3: Changes in macro F1 values with varying percentages of training data for the PG module.

On the other hand, for the other categories, it could be found that the model can use data efficiently and obtain the best result by training on only 1% samples.

More samples might help the model on some question categories, but more training time is consumed. The training time of the REINFORCE module is plotted in Figure 4. As can be seen, there is a significant increase in training time when training data increases from 1.0% to 1.2%. Together with the trend of the macro F1 value, as shown in Figure 3, we can empirically determine the best trade-off between model performance and training efficiency at 1.0%.

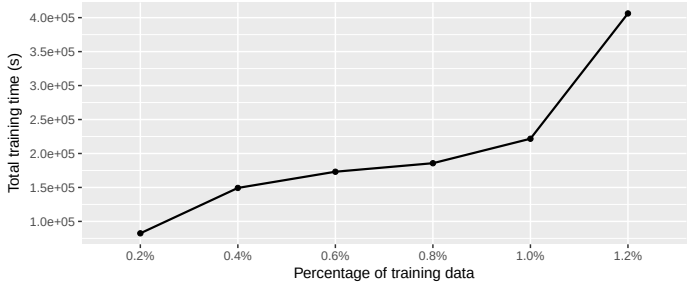

Figure 4: Changes in training time (in seconds) with varying percentages of training data.

5. Qualitative Analysis

In this section, we analyze the quality of our NS-CQA model in more detail. We first present some success cases where NS-CQA can predict the action sequence that produces correct answers. A detailed analysis of typical errors is then performed, which sheds light on the areas that can be further investigated.

5.1. Sample Cases

In Table 6 below, we present some example questions from different categories that our model NS-CQA can correctly predict action sequences.

We can inspect the complexity of the CQA problem from these instances. As is demonstrated in the table, Simple and Logical questions are simplest to answer since commonly, only 2-3 actions are needed. The following four categories, i.e., Quantitative, Comparative, Verification, and Quantitative Count, are relatively more difficult types with around 3-4 operations. For instance, the Verification question "Is Alda Pereira-Lemaitre a citizen of France and Emmelsbüll-Horsbüll?" has an answer "YES and NO respectively". Answering this question involves selecting all countries which Alda Pereira-Lemaitre is a citizen of, and verifying whether France and Emmelsbüll-Horsbüll is in this set respectively. The last type, Comparative Count, is the most complex for questions of which will be transformed into more than five actions.

Evident from the Quantitative Count and the Comparative Count questions in the last two rows of the table, answering CQA questions involve discrete actions. In the case of the question "How many assemblies or courts have control over the jurisdiction of the Free Hanseatic City of Bremen?", the set operation Union is required. In the case of the question "How many art genres express more number of humen or concepts than floral painting?", numerical operations (GreaterThan and Count) are required.

These example questions attest to the challenging nature of the CQA dataset and the capability of our NS-CQA model.

5.2. Error Analysis

To analyze the limitations of our NS-CQA model, 200 samples in each category that produce incorrect answers are randomly selected from the test dataset. In summary, a large number of errors can be categorised into one of the following five classes.

5.2.1. Linking Problem

Since different entities/types might have the same surface name, in addition to literal similarity, the embedding of the description of entities/types and the embedding of question is employed to compute semantic similarity in our approach. When mapping the predicates to queries, a state-of-the-art convolutional sequence to sequence (Seq2Seq) learning model [40] implemented in fairSeq [43] is used. Even so, some linking problems remain.

Example:. "Where are around the same number of geographic locations located on as Big Salmon Range?".

When our model is answering the above question, the relation 'located on street' is wrongly linked to the question instead of the correct relation 'located on terrain feature'. This type of errors can be addressed by learning better semantic meaning of the entities/types/relations from the context in the knowledge graph.

5.2.2. Infeasible Action

NS-CQA occasionally produces meaningless and repetitive actions which are semantically incorrect. For instance, some actions are predicted to union the same set, which is reluctant. In some cases, two repeated 'Count' actions are predicted.

Table 6: Examples of action sequences correctly predicted by NS-CQA for different types of questions.

Q. type	Question	KB artifacts	Action sequence	Answer
Simple	Which administrative territory is Danilo Ribeiro an inhabitant of?	**E1:** Danilo Ribeiro **R1:** country of citizenship **T1:** administrative territory	Select(E1, R1, T1) EOQ	Brazil
Logical	Which administrative territories are twin towns of London but not Bern?	**E1:** London **E2:** Bern **R1:** twinned adm. body **T1:** administrative territory	Select(E1, R1, T1) Diff(E2, R1, T1) EOQ	Sylhet, Tokyo, Podgorica, Phnom Penh, Delhi, Los Angeles, Sofia, New Delhi, . . .
Quantitative	Which sports teams have min number of stadia or architectural structures as their home venue?	**R1:** home venue **T1:** sports team **T2:** stadium **T3:** architectural structure	SelectAll(T1, R1, T2) SelectAll(T1, R1, T3) ArgMin() EOQ	Detroit Tigers, Drbak-Frogn IL, Club Sport Emelec, Chunichi Dragons, . . .
Comparative	Which buildings are a part of lesser number of architectural structures and universities than Midtown Tower?	**E1:** Midtown Tower **R1:** part of **T1:** building **T2:** architectural structure **T3:** university	SelectAll(T1, R1, T2) SelectAll(T1, R1, T3) LessThan(E1) EOQ	Amsterdam Centraal, Hospital de Sant Pau, Budapest Western Railway Terminal, El Castillo, . . .
Verification	Is Alda Pereira-Lemaitre a citizen of France and Emmelsbüll-Horsbüll?	**E1:** Alda Pereira-Lemaitre **E2:** France **E3:** Emmelsbüll-Horsbüll **R1:** country of citizenship **T1:** administrative territory	Select(E1, R1, T1) Bool(E2) Bool(E3) EOQ	YES and NO respectively
Quantitative Count	How many assemblies or courts have control over the jurisdiction of Free Hanseatic City of Bremen?	**E1:** Bremen **R1:** applies to jurisdiction **T1:** deliberative assembly **T2:** court	Select(E1, R1, T1) Union(E1, R1, T2) Count() EOQ	2
Comparative Count	How many art genres express more number of humen or concepts than floral painting?	**E1:** floral painting **R1:** depicts **T1:** art genre **T2:** human **T3:** concept	SelectAll(T1, R1, T2) SelectAll(T1, R1, T3) GreaterThan(E1) Count() EOQ	8

Example:. "What social groups had Canada and Austria as their member?"

When our model is solving the above question, it predicts the following action sequence:

Select (Canada, member of, social group)
Bool (Austria)
EOQ

The operator 'Bool' is invalid since in this question the expected answer type is entities but not Boolean values. Semantic-based constraints could be employed to make the model produce feasible actions.

5.2.3. Spurious Problem

In our approach, the pseudo-gold action sequences are generated by a BFS algorithm. Therefore corresponding to each question, multiple possible sequences may evaluate to the same expected results. Among these sequences, there might be some spurious action sequences. When training the model with such action sequences, the model may be misled and produce incorrect actions.

Example:. "Which musical ensembles were formed at Belfast?"

Our model transforms the above question into the following action sequence:

Select(Belfast, location, musical ensemble)
Inter(Belfast, location, musical ensemble)
EOQ

The second action 'Inter' is unnecessary to this question. Rule-based constraints could be incorporated to restrict the search process to meaningful actions.

5.2.4. Order of Arguments

To decide the order of the entities/types in the actions is a difficult problem. For actions 'Select', 'Inter', 'Diff', and 'Union', the order of the arguments is decided by the following rule: the first argument in a triple pattern is related to the entity, and the last argument is associated with the type. In most cases, a sequence of entities/types in actions follows the order they appear in the question. Though our model is also trained to handle the situation that the sequence of entities/types does not appear in the same order, in some cases, the model is confused about which order to follow.

Example:. "Is Bernhard II, Duke of Saxe-Jena a child of William, Duke of Saxe-Weimar?"

Our model transforms the above question into the following action sequence:

Select (Bernhard II. Duke of Saxe-Jena, child, common name)

Bool (William. Duke of Saxe-Weimar)
EOQ

However, the correct action sequence should be the following. As can be seen, the order of the two entities is wrong in the predicted sequence.

Select (William. Duke of Saxe-Weimar, child, common name)
Bool (Bernhard II. Duke of Saxe-Jena)
EOQ

In future work, we will investigate whether incorporating more positional information can help alleviate this problem.

5.2.5. Approximation-related Problem

The action 'Almost' is used to find the set of entities whose number is approximately the same as a given value, and such operation appears in the following four categories of questions: Quantitative Reasoning, Quantitative Count, Comparative Reasoning, and Comparative Count. The questions involving the 'Almost' action account for 4% of the total test dataset. When solving such questions, the range of the approximate interval is naturally vague. We define the following ad-hoc rule to address this vagueness: suppose we are required to find the value around N, when N is no larger than 5, the interval is $[N-1, N+1]$; when N is more significant than 5, the range is $[N-5, N+5]$. In some cases, this rule works, but in others not.

Example:. "Which political territories have diplomatic relations with approximately 14 administrative territories?"

The following action sequence could be produced to solve such questions:

SelectAll (political territorial entity, diplomatic relation, administrative territorial entity)
Almost (14)
EOQ

Following our rule, the approximate interval here should be [9, 19]. However, the correct answer (political territorial entities) may have several administrative territories outside this range. Thus, the unfixed approximate interval may impair the performance of our model. We can manually tweak the rule of deciding the approximate interval. However, we emphasize that our model aims to show a robust framework to solve complex questions, but not to guess rules for approximation.

6. Conclusion

Answering complex questions on KBs is a challenging problem as it requires a model to perform discrete operations over KBs. State-of-the-art techniques combine neural networks and symbolic execution to address this problem. While practical, the challenges of these techniques reside in data-inefficiency, reward sparsity, and ample search space.

In this paper, we propose a data-efficient neural-symbolic model for complex KBQA that combines simple yet effective techniques, addressing some of the above deficiencies.

Firstly, we augment the model with a memory buffer. When the memory buffer maintains the generated successful trials for each training question, it will guide the model to replay and resample the promising trials more frequently, thus mitigating the data-inefficiency problem.

Secondly, by comparing the generated trials with the trials stored in the memory, we assign a bonus to the reward, which is the combination of proximity and novelty. Also, we propose an adaptive reward function. The reward bonus and the adaptive reward reshape the sparse reward into dense feedback that can efficiently guide policy optimization. Employing the curriculum-learning scheme, we gradually increase the proportion of proximity while decreasing the weight of novelty. By doing this, we encourage the model to find new trials while remembering the past successful trials.

Thirdly, we incorporate the copy and masking mechanisms in the model, and carefully design a set of primitive actions, to drastically reduce the size of the decoder output vocabulary by orders of magnitude. This significant reduction improves not only training efficiency but also model generalizability. Also, our actions free the model from the need to maintain complex intermediate memory modules, thus simplifies network design.

We conduct experiments on two challenging datasets on complex question answering. In comparison with three state-of-the-art techniques, our model achieves the best performance and significantly outperforming them in both the datasets.

Acknowledgements

This work was partially supported by the National Key Research and Development Program of China under grants (2018YFC0830200), the Natural Science Foundation of China grants (U1736204, 61602259), the Judicial Big Data Research Centre, School of Law at Southeast University, and the project no. 31511120201 and 31510040201.

References

[1] J. Berant, A. Chou, R. Frostig, P. Liang, Semantic parsing on freebase from question-answer pairs, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 1533–1544, 2013.

[2] X. Yao, B. Van Durme, Information extraction over structured data: Question answering with freebase, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 956–966, 2014.

[3] W.-t. Yih, X. He, C. Meek, Semantic parsing for single-relation question answering, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), vol. 2, 643–648, 2014.

[4] A. Bordes, N. Usunier, S. Chopra, J. Weston, Large-scale simple question answering with memory networks, arXiv preprint arXiv:1506.02075 .

[5] W.-t. Yih, M.-W. Chang, X. He, J. Gao, Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), vol. 1, 1321–1331, 2015.

[6] A. Saha, V. Pahuja, M. M. Khapra, K. Sankaranarayanan, S. Chandar, Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[7] A. Saha, G. A. Ansari, A. Laddha, K. Sankaranarayanan, S. Chakrabarti, Complex Program Induction for Querying Knowledge Bases in the Absence of Gold Programs, Transactions of the Association for Computational Linguistics 7 (2019) 185–200.

[8] C. Liang, J. Berant, Q. Le, K. D. Forbus, N. Lao, Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 23–33, 2017.

[9] G. A. Ansari, A. Saha, V. Kumar, M. Bhambhani, K. Sankaranarayanan, S. Chakrabarti, Neural program induction for KBQA without gold programs or query annotations, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, AAAI Press, 4890–4896, 2019.

[10] N. Savinov, A. Raichuk, D. Vincent, R. Marinier, M. Pollefeys, T. P. Lillicrap, S. Gelly, Episodic Curiosity through Reachability, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, URL https://openreview.net/forum?id=SkeK3s0qKQ, 2019.

[11] C. Liang, M. Norouzi, J. Berant, Q. V. Le, N. Lao, Memory augmented policy optimization for program synthesis and semantic parsing, in: Advances in Neural Information Processing Systems, 9994–10006, 2018.

[12] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (10) (2014) 78–85.

[13] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 1247–1250, 2008.

[14] J. Gu, Z. Lu, H. Li, V. O. Li, Incorporating Copying Mechanism in Sequence-to-Sequence Learning, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 1631–1640, 2016.

[15] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning (ICML'09), 41–48, 2009.

[16] M. Fang, T. Zhou, Y. Du, L. Han, Z. Zhang, Curriculum-guided Hindsight Experience Replay, in: Advances in Neural Information Processing Systems, 12602–12613, 2019.

[17] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, J. Suh, 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), vol. 2, 201–206, 2016.

[18] J. Lehmann, T. Furche, G. Grasso, A.-C. N. Ngomo, C. Schallhart, A. Sellers, C. Unger, L. Buhmann, D. Gerber, K. Hoffner, et al., DEQA: deep web extraction for question answering, in: International Semantic Web Conference, Springer, 131–147, 2012.

[19] J. Bao, N. Duan, M. Zhou, T. Zhao, Knowledge-based question answering as machine translation, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 967–976, 2014.

[20] S. Hu, L. Zou, X. Zhang, A State-transition Framework to Answer Complex Questions over Knowledge Base, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2098–2108, 2018.

[21] L. Dong, F. Wei, M. Zhou, K. Xu, Question answering over freebase with multi-column convolutional neural networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), vol. 1, 260–269, 2015.

[22] X. He, D. Golub, Character-Level Question Answering with Attention, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 1598–1607, 2016.

[23] D. Lukovnikov, A. Fischer, J. Lehmann, S. Auer, Neural network-based question answering over knowledge graphs on word and character level, in: Proceedings of the 26th international conference on World Wide Web, International World Wide Web Conferences Steering Committee, 1211–1220, 2017.

[24] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, R. Socher, Ask me anything: Dynamic memory networks for natural language processing, in: International Conference on Machine Learning, 1378–1387, 2016.

[25] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, J. Weston, Key-Value Memory Networks for Directly Reading Documents, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 1400–1409, 2016.

[26] K. Xu, Y. Lai, Y. Feng, Z. Wang, Enhancing Key-Value Memory Neural Networks for Knowledge Based Question Answering, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2937–2947, 2019.

[27] K. Luo, F. Lin, X. Luo, K. Zhu, Knowledge Base Question Answering via Encoding of Complex Query Graphs, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2185–2194, 2018.

[28] D. Guo, D. Tang, N. Duan, M. Zhou, J. Yin, Dialog-to-Action: Conversational Question Answering Over a Large-Scale Knowledge Base, in: Advances in Neural Information Processing Systems, 2946–2955, 2018.

[29] S. Reddy, M. Lapata, M. Steedman, Large-scale semantic parsing without question-answer pairs, Transactions of the Association for Computational Linguistics 2 (2014) 377–392.

[30] A. Bordes, J. Weston, N. Usunier, Open question answering with weakly supervised embedding models, in: Joint European conference on machine learning and knowledge discovery in databases, Springer, 165–180, 2014.

[31] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, arXiv preprint arXiv:1410.5401 .

[32] W. Zaremba, I. Sutskever, Reinforcement learning neural turing machines-revised, arXiv preprint arXiv:1505.00521 .

[33] A. Neelakantan, Q. V. Le, I. Sutskever, Neural programmer: Inducing latent programs with gradient descent, arXiv preprint arXiv:1511.04834 .

[34] P. Pasupat, P. Liang, Compositional Semantic Parsing on Semi-Structured Tables, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), vol. 1, 1470–1480, 2015.

[35] K. Guu, P. Pasupat, E. Liu, P. Liang, From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 1051–1062, 2017.

[36] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, J. Tenenbaum, Neural-symbolic vqa: Disentangling reasoning from vision and language understanding, in: Advances in Neural Information Processing Systems, 1039–1050, 2018.

[37] A. Saha, G. A. Ansari, A. Laddha, K. Sankaranarayanan, S. Chakrabarti, Complex Program Induction for Querying Knowledge Bases in the Absence of Gold Programs, Transactions of the Association for Computational Linguistics 7 (2019) 185–200.

[38] Z. Huang, W. Xu, K. Yu, Bidirectional LSTM-CRF models for sequence tagging, arXiv preprint arXiv:1508.01991 .

[39] W. Yin, M. Yu, B. Xiang, B. Zhou, H. Schütze, Simple Question Answering by Attentive Convolutional Neural Network, in: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 1746–1756, 2016.

[40] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y. N. Dauphin, Convolutional sequence to sequence learning, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 1243–1252, 2017.

[41] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8 (3-4) (1992) 229–256.

[42] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, V. Goel, Self-critical sequence training for image captioning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 7008–7024, 2017.

[43] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, M. Auli, fairseq: A Fast, Extensible Toolkit for Sequence Modeling, in: Proceedings of NAACL-HLT 2019: Demonstrations, 2019.