Few-Shot Complex Knowledge Base Question Answering via Meta Reinforcement Learning

Yuncheng Hua[†,§], Yuan-Fang Li[◇], Gholamreza Haffari[◇], Guilin Qi[†,‡*] and Tongtong Wu[†]
[†]School of Computer Science and Engineering, Southeast University, China
[◇]Faculty of Information Technology, Monash University, Australia
[§]Southeast University-Monash University Joint Research Institute, China
[‡]Key Laboratory of Computer Network and Information Integration, Southeast University
[†]{devinhua, gqi, wutong8023}@seu.edu.cn
[◇]{yuanfang.li, gholamreza.haffari}@monash.edu

Abstract

Complex question-answering (CQA) involves answering complex natural-language questions on a knowledge base (KB). However, the conventional neural program induction (NPI) approach exhibits uneven performance when the questions have different types, harboring inherently different characteristics, e.g., difficulty level. This paper proposes a meta-reinforcement learning approach to program induction in CQA to tackle the potential distributional bias in questions. Our method quickly and effectively adapts the meta-learned programmer to new questions based on the most similar questions retrieved from the training data. The meta-learned policy is then used to learn a good programming policy, utilizing the *trial* trajectories and their rewards for similar questions in the support set. Our method achieves state-of-the-art performance on the CQA dataset (Saha et al., 2018) while using only five trial trajectories for the top-5 retrieved questions in each support set, and meta-training on tasks constructed from only 1% of the training set. We have released our code at https://github.com/DevinJake/MRL-CQA.

1 Introduction

Knowledge-base question-answering (KBQA) interrogates a knowledge-base (KB) (Yin et al., 2016; Yu et al., 2017; Jin et al., 2019) by interpreting natural-language questions as logical forms (*annotations*), which can be directly executed on the KB to yield answers (*denotations*) (Pasupat and Liang, 2016). KBQA includes *simple questions* that retrieve answers from single-hop triples ("what is Donald Trump's nationality") (Berant et al., 2013; Yih et al., 2014), *multi-hop questions* that infer answers over triple chains of at least 2 hops under specific constraints ("who is the president of the European Union 2012") (Yih et al., 2016; Liang et al., 2017), and *complex questions* that involve set operations ("how many rivers flow through India and China") (Saha et al., 2019). In particular, complex question answering (CQA) (Saha et al., 2018) is a sophisticated KBQA task in which a sequence of discrete *actions*—e.g., set intersection and union, counting, comparison—needs to be executed, and is the subject of this paper.

Consider the complex question "How many rivers flow through India and China?". We first form a set of entities whose type is river and flow in China from the KB. We then form another set for rivers that flow through India. The answer is then generated by *counting* the entities in the *intersection* of the two sets. More concretely, the question is transformed into the action sequence "Select (China, flow, river), Intersection (India, flow, river), Count", which is executed on the KB to yield the answer. As such, the CQA task results in a massive search space beyond just entities in the KB and includes (lists of) Boolean values and integers. Multi-hop questions only require the join operator. In contrast, CQA requires various types of additional symbolic reasoning, e.g., logical, comparative, and quantitative reasoning (Shen et al., 2019; Ansari et al., 2019), where a more diverse array of complex queries is involved (Saha et al., 2019). The massive search space and complex queries make CQA considerably challenging and more complicated than multi-hop question answering.

Due to the difficulty of collecting annotations, the existing CQA dataset (Saha et al., 2018) only contains the denotations for each question. The literature takes two approaches to deal with the missing annotations. The first approach aims to transform learning a CQA model into learning by demonstration, aka imitation learning, where a *pseudo-gold* action sequence is produced for the questions in the training set (Guo et al., 2018). This is done by employing a blind search algorithm, i.e.,

[*]Corresponding Author.

breadth-first search (BFS), to find a sequence of actions whose execution would yield the correct answer. This pseudo-gold annotation is then used to train the programmer using teacher forcing, aka behaviour cloning. However, BFS inevitably produces a single annotation and is ignorant to many other plausible annotations yielding the correct answer. To alleviate this issue, a second approach was proposed based on reinforcement learning (RL) to use the search policy prescribed by the programmer (Hua et al., 2020; Neelakantan et al., 2016; Liang et al., 2017). Compared to BFS which is a blind search algorithm, the RL-trained programmer can be regarded as an informed search algorithm for target programs. Therefore, the RL policy not only addresses the limitation of the 1-to-1 mapping between the questions and annotations, but also produces reasonable programs faster than BFS.

The conventional approach to CQA is to train *one* model to fit the entire training set, and then use it for answering all complex questions at the test time. However, such a one-size-fits-all strategy is sub-optimal as the test questions may have diversity due to their inherently different characteristics (Huang et al., 2018). For instance, in the CQA dataset, the samples could be categorized into seven different types, e.g., those capturing logical/comparative/quantitative reasoning. The length and complexity of questions in one group are likely to differ from those in other groups. Therefore, action sequences relevant to different groups may have significant deviations, and it is hard to learn a one-size-fits-all model that could adapt to varied types of questions. An exception is (Guo et al., 2019), which proposes a few-shot learning approach, i.e., S2A, to solve the CQA problem with a retriever and a meta-learner. The retriever selects similar instances from the training dataset to form *tasks*, and the meta-learner is trained on these tasks to learn how to quickly adapt to a new task created by the target question of interest at the test time. However, Guo et al. (2019) make use of teacher forcing within the learning by demonstration approach, which suffers from the aforementioned drawbacks. Also, though S2A is the most similar to ours, the tasks are very different. S2A aims to answer *context-dependent* questions, where each question is part of a *multiple-turn* conversation. On the contrary, we consider the different task where the questions are *single-turn* and have no context. Thus, a novel challenge arises in retrieving accurate support sets without conversation-based context information.

In this paper, we propose a Meta-RL approach for CQA (MRL-CQA), where the model adapts to the target question by *trials* and the corresponding reward signals on the retrieved instances. In the meta-learning stage, our approach learns a RL policy across the tasks for both (i) collecting trial trajectories for effective learning, and (ii) learning to adapt programmer by effectively combining the collected trajectories.

The accumulated general knowledge acquired during meta-learning enables the model to generalize over varied tasks instead of fitting the distribution of data points from a single task. Thus, the tasks generated from tiny (less than 1%) portion of the training data are sufficient for meta learner to acquire the general knowledge. Our method achieves state-of-the-art performance on the CQA dataset with overall macro and micro F1 scores of 66.25% and 77.71%, respectively.

2 Meta-RL for Complex Question Answering

The problem we study in this paper is transforming a complex natural-language question into a sequence of actions, i.e., a sequence-to-sequence learning task. By executing the actions, relevant triples are fetched from the KB, from which the answer to the question is induced. We tackle this problem with few-shot meta reinforcement learning to decrease the reliance on data annotation and increase the accuracy for different questions.

Let $\boldsymbol{q}$ denote the input sequence, including the complex question and the KB artifacts, i.e., entities, relations, and types in KB that are relevant to the problem. Let $\boldsymbol{\tau}$ denotes the output sequence, i.e., an action sequence that the agent generates to answer the question. Let $R(\boldsymbol{\tau}|\boldsymbol{q}) \in [0,1]$ denotes the partial reward feedback that tells whether or not the action sequence yields the correct answer. To simplify the notation, we denote the reward function by $R(\boldsymbol{\tau})$. The training objective is to maximize the expected reward by optimizing the parameter $\boldsymbol{\theta}$ of the policy $\pi(\boldsymbol{\tau}|\boldsymbol{q};\boldsymbol{\theta})$, i.e., improving the accuracy of the policy in answering unseen questions. For the test, the agent needs to generate an action sequence $\boldsymbol{\tau}^*$ for the input sequence using a search algorithm, e.g., greedy decoding, which is then executed on KB to get the answer.

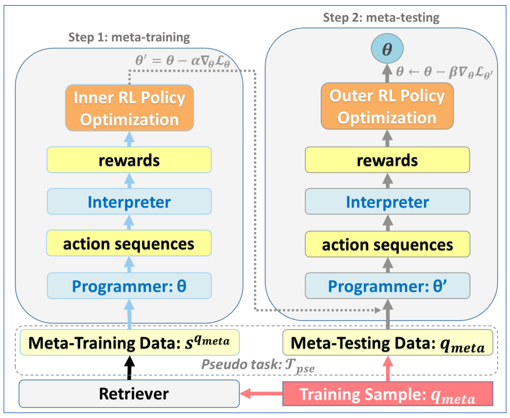
Step 1: meta-training
$\theta' = \theta - \alpha\nabla_\theta\mathcal{L}_\theta$
Inner RL Policy Optimization
rewards
Interpreter
action sequences
Programmer: θ
Step 2: meta-testing
θ
$\theta \leftarrow \theta - \beta\nabla_\theta\mathcal{L}_{\theta'}$
Outer RL Policy Optimization
rewards
Interpreter
action sequences
Programmer: θ'
Meta-Training Data: $s^{q_{meta}}$
Meta-Testing Data: q_{meta}
Pseudo task: $\mathcal{T}_{pse}$
Retriever
Training Sample: q_{meta}

Figure 1: The high-level architecture of our approach.

2.1 Overview of the Framework

Our framework for few-shot learning of CQA is illustrated in Figure 1. In our framework, we view each new training question as the test sample of a pseudo task, and we aim to learn a specific model devoted to solving the task. When faced with a question $\boldsymbol{q_{meta}}$, we first employ the retriever to find top-N samples $\boldsymbol{s^{q_{meta}}}$ in the training dataset, which are the most similar to $\boldsymbol{q_{meta}}$. We consider $\boldsymbol{s^{q_{meta}}}$ as meta-training data used to learn a particular model, and view the question $\boldsymbol{q_{meta}}$ as the meta-testing data to evaluate the model. Therefore, meta-training data $\boldsymbol{s^{q_{meta}}}$ and meta-testing data $\boldsymbol{q_{meta}}$ form a pseudo task $\boldsymbol{\mathcal{T}_{pse}}$.

In the meta-training step (Step 1 in Figure 1), the action sequences that correspond to $\boldsymbol{s^{q_{meta}}}$ will be generated based on the current parameter $\boldsymbol{\theta}$ of the *programmer*. The *interpreter* executes the action sequences and evaluates the generated answers to produce rewards. The rewards lead to gradient updates that finetune the current model to get a task-specific *programmer* with the parameter of $\boldsymbol{\theta'}$. After that, in the meta-testing step (Step 2 in Figure 1), the actions of $\boldsymbol{q_{meta}}$ are produced based on $\boldsymbol{\theta'}$ and are evaluated to update $\boldsymbol{\theta}$. The training approach is depicted in Algorithm 1. In both the meta-training and meta-testing steps, REINFORCE (Williams, 1992) is used to optimize the *programmer*.

Similarly, in the inference phase, we consider each test question as a new individual task. We retrieve top-N data points from the training dataset to form the meta-training data. Instead of applying the general *programmer* with $\boldsymbol{\theta}$ directly, the meta-training data is used to finetune a specific parameter $\boldsymbol{\theta'}$ that fits the test question and infer the output.

2.2 Programmer and Interpreter

Programmer Our *programmer* is a sequence-to-sequence (Seq2Seq) model. Given the input sequence $\boldsymbol{q}$ with tokens $(w_1, \ldots, w_M)$, the *programmer* produces actions $(a_1, \ldots, a_T)$. The input sequence is the original complex question concatenated with the KB artifacts appear in the query, and the output is the words or tokens. The output at each time step is a single token.

In the *programmer*, the encoder is a Long Short Term Memory (LSTM) network that takes a question of variable length as input and generates an encoder output vector $\boldsymbol{e}_i$ at each time step i as: $(\boldsymbol{e}_i, \boldsymbol{h}_i) = LSTM[\phi_E(w_i), \boldsymbol{h}_{i-1}]$. Here $\phi_E(w_i)$ is word embedding of token w_i, and $(\boldsymbol{e}_i, \boldsymbol{h}_i)$ is the output and hidden vector of the i-th time step. The dimension of $\boldsymbol{e}_i$ and $\boldsymbol{h}_i$ are set as the same.

Our decoder of the *programmer* is another attention-based LSTM model that selects output token a_t from the output vocabulary $\mathcal{V}_{output}$. The decoder generates a hidden vector $\boldsymbol{g}_t$ from the previous output token a_{t-1}. The previous step's hidden vector $\boldsymbol{g}_{t-1}$ is fed to an attention layer to obtain a context vector $\boldsymbol{c}_t$ as a weighted sum of the encoded states using the standard attention mechanism. The current step's $\boldsymbol{g}_t$ is generated via $\boldsymbol{g}_t = LSTM\{\boldsymbol{g}_{t-1}, [\phi_D(a_{t-1}), \boldsymbol{c}_t]\}$, where ϕ_D is the word embedding of input token a_{t-1}. The decoder state $\boldsymbol{g}_t$ is used to compute the score of the target word $v \in \mathcal{V}_{output}$ as,

$$\pi(a_t = v|\boldsymbol{a}_{<t}, \boldsymbol{q}) = \text{softmax}(\mathbf{W} \cdot \boldsymbol{g}_t + \boldsymbol{b})_v \qquad (1)$$

where $\mathbf{W}$ and $\boldsymbol{b}$ are trainable parameters, and $\boldsymbol{a}_{<t}$ denotes all tokens generated before the time step t. We view all the weights in the *programmer* as the parameter $\boldsymbol{\theta}$, thus we have the probability that the *programmer* produces an action sequence $\boldsymbol{\tau}$ with tokens $\{a_1, ..., a_T\}$ as,

$$\pi(\boldsymbol{\tau}|\boldsymbol{q}; \theta) = \prod_{t=1}^{T} \pi(a_t = v|\boldsymbol{a}_{<t}, \boldsymbol{q}). \qquad (2)$$

When adapting the policy to a target question, our *programmer* outputs action sequences following the distribution computed by equation 2. By treating decoding as a stochastic process, the *programmer* performs random sampling from the probability distribution of action sequences to increase the output sequences' diversity.

Interpreter After the *programmer* generates the entire sequence of actions, the *interpreter* executes the sequence to produce an answer. It compares the predicted answer with the ground-truth answer and outputs a partial reward. If the type of the output answer is different from that of the ground-truth answer, the action sequence that generates this answer will be given a reward of 0. Otherwise, to alleviate the sparse reward problem, the *interpreter* takes the Jaccard score of the output answer set and the ground-truth answer set as the partial reward, and sends it back to update parameters of the *programmer* as the supervision signal.

2.3 Meta Training and Testing

We formulate training of the programmer in a RL setting, where an agent interacts with an environment in discrete time steps. At each time step t, the agent produces an action (in our case a word/token) a_t sampled from the policy $\pi(a_t|\boldsymbol{a}_{<t}, \boldsymbol{q}; \boldsymbol{\theta})$, where $\boldsymbol{a}_{<t}$ denotes the sequence generated by the agent from step 1 to $t-1$, and $\boldsymbol{q}$ is the input sequence. The policy of the agent is the *programmer*, i.e., LSTM-attention model M with parameter $\boldsymbol{\theta}$. The natural-language question concatenated with the KB artifacts will be fed into the encoder as an input, and a sequence of actions is output from the decoder. In our work, we regard each action sequence produced by the model as one trajectory. The action sequence is therefore executed to yield a generated answer, and the similarity between the output answer with the ground-truth answer is then computed. The environment considers the similarity as the reward R corresponding to the trajectory $\boldsymbol{\tau}$ and gives it back to the agent. In standard RL, the parameter of the policy $\boldsymbol{\theta}$ is updated to maximize the expected reward, $\mathbb{E}_{\boldsymbol{\tau}\sim\pi(\boldsymbol{\tau}|\boldsymbol{q};\boldsymbol{\theta})}[R(\boldsymbol{\tau})]$.

In our work, answering each question in the training dataset is considered as an individual task, and a model adaptive to a new task is learned from the support set questions. To make the meta-learned model generalize to all unseen tasks, we sample the tasks following the distribution of tasks in the training dataset. We first sample a small subset of the questions Q_{meta} from the training dataset and expand the questions into tasks $\mathcal{T}_{meta}$ through retrieving the top-N samples, then extract a batch of tasks $\mathcal{T}'$ from $\mathcal{T}_{meta}$ under the distribution of tasks in $\mathcal{T}_{meta}$ to update parameters.

To fully use the training dataset and decrease training time, we study how to train a competitive model by using as few training samples as possible. As we view CQA as a RL problem under few-shot learning conditions, we make use of Meta-RL techniques (Finn et al., 2017) to adapt the programmer to a new task with a few training samples. Meta-RL aims to meta-learn an agent that can rapidly learn the optimal policy for a new task $\mathcal{T}$. It amounts to learn optimized $\boldsymbol{\theta}^*$ using K trial trajectories and the rewards for the support set of a new task.

We use the gradient-based meta-learning method to solve the Meta-RL problem such that we can obtain the optimal policy for a given task after performing a few steps of vanilla policy gradient (VPG) (Williams, 1992; Sutton et al., 2000). We divide the meta-learning process into two steps to solve a task, namely the meta-training step and the meta-testing step. Suppose we are trying to solve the pseudo-task $\mathcal{T}_{pse}$, which consists of N meta-training questions $\boldsymbol{s}^{\boldsymbol{q}_{meta}}$ that are the most similar to the meta-testing sample q_{meta}. The model first generates K trajectories for each question in $\boldsymbol{s}^{\boldsymbol{q}_{meta}}$ based on $\boldsymbol{\theta}$. The reward of each trajectory is given by the environment and then subsequently used to compute $\boldsymbol{\theta}'$ adapted to task $\mathcal{T}_{pse}$, as

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \eta_1 \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{q}\in\boldsymbol{s}^{\boldsymbol{q}_{meta}}} \mathbb{E}_{\boldsymbol{\tau}\sim\pi(\boldsymbol{\tau}|\boldsymbol{q};\theta)}[R(\boldsymbol{\tau})] \quad (3)$$

During meta-testing, another K' trajectories corresponding to question $\boldsymbol{q}_{meta}$ are further produced by $\boldsymbol{\theta}'$. The reward of K' trajectories are considered as the evaluation of the adapted policy $\boldsymbol{\theta}'$ for the given task $\mathcal{T}_{pse}$; thus we have the objective,

$$J(\boldsymbol{\theta}') \overset{\text{def}}{=} \mathbb{E}_{\boldsymbol{\tau}'\sim\pi(\boldsymbol{\tau}'|\boldsymbol{q}_{meta};\boldsymbol{\theta}')}[R(\boldsymbol{\tau}')] \quad (4)$$

The parameter of the generic policy $\boldsymbol{\theta}$ are then trained by maximising the objective $J(\boldsymbol{\theta}')$,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_2 \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}') \quad (5)$$

In each VPG step, since we have N samples in $\boldsymbol{s}^{\boldsymbol{q}_{meta}}$, we use N policy gradient adaptation steps to update $\boldsymbol{\theta}'$. Meanwhile, we use one policy gradient step to optimize θ based on the evaluation of $\boldsymbol{\theta}'$. Monte Carlo integration is used as the approximation strategy in VPG (Guu et al., 2017). We summarise the meta-learning approach in Alg.1.

When making inferences, for each question $\boldsymbol{q}_{test}$, the retriever creates a pseudo-task, similar to the meta-learning process. The top-N similar questions to $\boldsymbol{q}_{test}$ form the support set $\boldsymbol{s}^{\boldsymbol{q}_{test}}$, and are used to obtain the adapted model $\boldsymbol{\theta}^{*'}$, starting from the meta learned policy $\boldsymbol{\theta}^*$. The adapted model is then used to generate the program and compute the target question's final answer.

```
Algorithm 1: Meta-RL (training time)
   Input: Dataset $Q_{train}$, step size $\eta_1$, $\eta_2$
   Output: Meta-learned policy $\boldsymbol{\theta}^*$
 1 Randomly initialize $\boldsymbol{\theta}$
 2 Randomly sample $Q_{meta} \sim Q_{train}$
 3 Expand $Q_{meta} \rightarrow \mathcal{T}_{meta}$
 4 while not done do
 5     Sample a batch of tasks $\mathcal{T}' \sim \mathcal{T}_{meta}$
 6     for $\mathcal{T}_{pse} \in \mathcal{T}'$ do
 7         $\mathcal{L} \leftarrow 0$
 8         for each question $\boldsymbol{q} \in \boldsymbol{s}^{\boldsymbol{q}_{meta}}$ do
 9             Sample $K$ trajectories:
                 $\boldsymbol{\tau}_k \sim \pi(\boldsymbol{\tau}|\boldsymbol{q};\boldsymbol{\theta})$
10             $\mathcal{L} \leftarrow$
                 $\mathcal{L} + \frac{1}{K}\sum_{k=1}^{K} R(\boldsymbol{\tau_k}) log p_{\boldsymbol{\theta}}(\boldsymbol{\tau_k})$
11         $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \eta_1 \nabla_{\boldsymbol{\theta}} \mathcal{L}$
12         Sample $K'$ trajectories:
             $\boldsymbol{\tau}_{k'} \sim \pi(\boldsymbol{\tau}|\boldsymbol{q}_{meta};\boldsymbol{\theta}')$
13         $J_{\boldsymbol{q}_{meta}}(\boldsymbol{\theta}') \leftarrow$
             $\frac{1}{K'}\sum_{k'=1}^{K'} R(\boldsymbol{\tau}_{k'}) log p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}_{k'})$
14     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_2 \nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_{pse}\in\mathcal{T}'} J_{\boldsymbol{q}_{meta}}(\boldsymbol{\theta}')$
15 Return The meta-learned policy $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
```

2.4 Question Retriever

We propose an unsupervised retriever that finds, from the training dataset, relevant support samples for the tasks in both the training and test phases. We propose a relevance function that measures the similarity between two questions in two aspects: (1) the number of KB artifacts (i.e., entities, relations, and types) in the questions and (2) question semantic similarity.

If the two questions have the same number of KB artifacts, the structure of their corresponding action sequences are more likely to be resembled. We calculate the similarity in terms of the number of entities of two questions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ by $sim_e(\boldsymbol{q}_1,\boldsymbol{q}_2) = 1 - \frac{|q_e(\boldsymbol{q}_1)-q_e(\boldsymbol{q}_2)|}{\max(q_e(\boldsymbol{q}_1),q_e(\boldsymbol{q}_2))}$. The function $q_e(\boldsymbol{q})$ counts the number of entities in the question. Similarly, we compute the similarities in terms of relations and types in the same way with $sim_r(\boldsymbol{q}_1,\boldsymbol{q}_2)$ and $sim_t(\boldsymbol{q}_1,\boldsymbol{q}_2)$ respectively. The KB artifact similarity $sim_a(\boldsymbol{q}_1,\boldsymbol{q}_2)$ is computed by the product of the above three similarities.

For two questions, the more common words they have, the more semantically similar they are. Based on this intuition, we propose a semantic similarity function based on Jaccard similarity in an unsupervised way. Suppose there is a set of i words $\{w_1^1,...,w_1^i\}$ in $\boldsymbol{q}_1$ and j words $\{w_2^1,...,w_2^j\}$ in $\boldsymbol{q}_2$, and word similarity $sim(w^i,w^j)$ is calculating using the Cosine similarity.

For each word in $\boldsymbol{q}_1$, we first collect the word pairs from the words in $\boldsymbol{q}_2$, whose highest similarity exceeds a pre-defined threshold value. We denote with $sem_{int}(\boldsymbol{q}_1,\boldsymbol{q}_2)$ the sum of similarity values of the word pairs:

$$sem_{int}(\boldsymbol{q}_1,\boldsymbol{q}_2) = \sum_{m=1}^{i}(\max_{n=1}^{j}(sim(w_1^m,w_2^n))) \tag{6}$$

After removing this set of highly similar words from the two questions, we denote the remaining tokens as $\{w_1^{remain}\}$ and $\{w_2^{remain}\}$, which represent the different parts of the two questions. We sum up the embeddings of the words in $\{w_1^{remain}\}$ as $\mathbf{w}_1^{remain}$, and compute $\mathbf{w}_2^{remain}$ in the same way. The function $sem_{diff}(\boldsymbol{q}_1,\boldsymbol{q}_2)$ measures how different $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are:

$$\begin{aligned} sem_{diff}(\boldsymbol{q}_1,\boldsymbol{q}_2) = \max(|\{w_1^{remain}\}|,|\{w_2^{remain}\}|) \\ *(1 - sim(\mathbf{w}_1^{remain},\mathbf{w}_2^{remain})), \end{aligned} \tag{7}$$

where $|\{w\}|$ returns the cardinality of the set $\{w\}$.

We define the semantic similarity between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ as: $sim_s(\boldsymbol{q}_1,\boldsymbol{q}_2) = \frac{sem_{int}(\boldsymbol{q}_1,\boldsymbol{q}_2)}{sem_{int}(\boldsymbol{q}_1,\boldsymbol{q}_2)+sem_{diff}(\boldsymbol{q}_1,\boldsymbol{q}_2)}$, and therefore calculate the similarity between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ with $sim_a(\boldsymbol{q}_1,\boldsymbol{q}_2) * sim_s(\boldsymbol{q}_1,\boldsymbol{q}_2)$.

3 Experiments

In this section, we present the empirical evaluation of our MRL-CQA framework.

Dataset. We evaluated our model on the large-scale CQA (Complex Question Answering) dataset (Saha et al., 2018). Generated from the Wikidata KB (Vrandecic and Krötzsch, 2014), CQA contains 944K/100K/156K QA pairs for training, validation, and testing, respectively. In the CQA dataset, each QA pair consists of a complex, natural-language question and the corresponding ground-truth answer (i.e., denotation). We note that annotations, i.e., gold action sequences related to the questions, are not given in the CQA dataset. The CQA questions are organized into seven categories of different characteristics, as shown in the Table 1. Some categories have entities as answers (e.g., "Simple Question"), while others have (lists of) numbers (e.g., "Quantitative (Count)") or Booleans (e.g., "Verification (Boolean)") as answers. The size of different categories in CQA is uneven. The number of instances in each category in the training set is 462K, 93K, 99K, 43K, 41K, 122K, and 42K for Simple Question, Logical Reasoning, Quantitative Reasoning, Verification (Boolean), Comparative Reasoning, Quantitative (Count), and Comparative (Count), respectively.

Based on the length of the induced programs and performance of the best models, we further organized the seven categories into two groups: *easy*—the first four categories, and *hard*—the last three types, in Table 1. We used the same evaluation metrics employed in the original paper (Saha et al., 2018), the F1 measure, to evaluate models.

Training Configuration. In the CQA dataset, since the annotated action sequence are not provided, we randomly sampled 1% of the training set (approx. 10K out of 944K training samples) and followed (Guo et al., 2019) to annotate them with pseudo-gold action sequences by using a BFS algorithm. We denoted the 1% questions and relevant pseudo-gold action sequences as Q_{pre}. The Q_{pre} was used to train the LSTM-based programmer, which was further optimized through the Policy Gradient (PG) algorithm (Williams, 1992; Sutton et al., 2000) with another 1% unannotated questions from the training set. We denoted this model by **PG**, which is also a model variant proposed in (Hua et al., 2020). We trained the meta learner on another 2K training samples (Q_{meta} in Alg.1), representing only approx. 0.2% of the training set. This meta learner is our full model: **MRL-CQA**.

In our work, we chose the attention-based LSTM model instead of the Transformer (Vaswani et al., 2017) to design the *programmer*. We set the sizes of embedding and hidden units in our LSTM model as 50 and 128 respectively, thus the maximum number of the parameters in our model is about 1.2M. However, the base model of the Transformer (12 layers, 12 heads, and 768 hidden units) has 110M parameters (Wolf et al., 2019), which are much more than those of our model. Given the small size of the training samples and the weak supervision signal (reward in our work), it is harder to train the model with more parameters. Therefore we chose LSTM rather than the Transformer.

We implemented our model in PyTorch and employed the Reptile meta-learning algorithm to optimize the meta-learned policy (Nichol and Schulman, 2018). The weights of the model and the word embeddings were randomly initialized and further updated within the process of training. In meta-learning, we set $\eta_1 = 1e-4$ (Equation 3) and $\eta_2 = 0.1$ (Equation 5). We set $N = 5$ and threshold value at 0.85 when forming the support set. For each question, we generate five action sequences to output the answers. The Adam optimizer is used in RL to maximizes the expected reward.

Among the baseline models, we ran the open-source code of KVmem (Saha et al., 2018) and CIPITR (Saha et al., 2019) to train the model. As the code of NSM (Liang et al., 2017) has not been made available, we re-implemented it and incorporated our programmer to predict programs, and employed the reinforcement learning settings in NSM to optimize the programmer. When inferring the testing samples, we used the top beam, i.e., the predicted program with the highest probability in the beam to yield the answers. We presented the best result we got to compare the baseline models.

3.1 Model Comparisons

We evaluated our model, MRL-CQA, against three baseline methods on the CQA dataset: KVmem, NSM, and CIPITR. It must be pointed out that CIPITR separately trained *one single model for each of the seven question categories*. We denote the model learned in this way as **CIP-Sep**. CIPITR also trained *one single model over all categories of training instances* and used this single model to answer all questions. We denote this single model as **CIP-All**. We separately present the performance of these two variations of CIPITR in Table 1. On the contrary, we tuned MRL-CQA on all categories of questions with one set of model parameters.

Table 1 summarizes the performance in F1 of the six models on the test set of CQA, organised into seven question categories. We note that the first four categories (first four rows in Table 1) are relatively simple, and the last three (middle three rows) are more challenging. We also report the overall macro and micro F1 values (last two rows).

As can be seen, our MRL-CQA model achieves the overall best macro and micro F1 values, achieving state-of-the-art results of 66.25% and 77.71%, respectively. MRL-CQA also achieves the best or second-best performance in six out of the seven categories. Of the three hardest categories (the last three types in Table 1), MRL-CQA delivers the best performance in all three types. This validates the effectiveness of our meta-learning-based approach in

Table 1: Performance comparison (measured in F1) of the seven methods on the CQA test set. For each category, best result is **bolded** and second-best result is underlined.

Question category	**KVmem**	**NSM**	**CIP-All**	**CIP-Sep**	**PG**	**MRL-CQA**
Simple Question	41.40%	88.83%	41.62%	**94.89%**	85.20%	88.37%
Logical Reasoning	37.56%	80.21%	21.31%	**85.33%**	78.23%	80.27%
Quantitative Reasoning	0.89%	36.68%	5.65%	33.27%	44.22%	**45.06%**
Verification (Boolean)	27.28%	58.06%	30.86%	61.39%	84.42%	**85.62%**
Comparative Reasoning	1.63%	59.45%	1.67%	9.60%	59.43%	**62.09%**
Quantitative (Count)	17.80%	58.14%	37.23%	48.40%	61.80%	**62.00%**
Comparative (Count)	9.60%	32.50%	0.36%	0.99%	38.53%	**40.33%**
Overall macro F1	19.45%	59.12%	19.82%	47.70%	64.55%	**66.25%**
Overall micro F1	31.18%	74.68%	31.52%	73.31%	75.40%	**77.71%**

effectively learning task-specific knowledge. Note that the two categories that MRL-CQA performs the best, Comparative Reasoning and Comparative (Count), both account for less than 5% of the training set, which further demonstrates our model's excellent adaptability.

Also, our RL-based programmer PG achieves second-best result in overall macro and micro F1, with about 2% difference below MRL-CQA. Moreover, PG achieves second-best in four categories. Such strong performance indicates the effectiveness of our CQA framework.

Besides the above main result, several important observations can be made from Table 1.

1. CIP-Sep got the best result in two categories, i.e., "Simple Question" and "Logical Reasoning". However, it performed poorly for the three hard categories. Consequently, the overall macro F1 value of CIP-Sep is substantially lower than both PG and MRL-CQA. Note that CIP-Sep trained a different model separately for each of the seven question categories. The results reported for each category were obtained from the models tuned specifically for each category (Saha et al., 2019), which necessitated a classifier to be trained first to recognize the question categories. Thus, CIP-Sep needs to re-train the models to adapt to new/changed categories, which impedes it from generalizing to unseen instances. However, we tuned our models on all questions with one set of model parameters, disregarding the question category information.

2. As presented in Table 1, CIP-All, the model that trained over all types of the questions, performed much worse in all the categories than CIP-Sep. A possible reason for CIP-All's significant performance degradation is that it is hard for such a one-size-fits-all model to find the weights that fit the training data when the examples vary widely. Besides, the imbalanced classes of questions also deteriorate the performance of the model. Different from CIPITR, MRL-CQA is designed to adapt appropriately to various categories of questions with one model, thus only needs to be trained once.

3. Our programmer and carefully-defined primitive actions presented in this work were used in our re-implementation of NSM. In the hard categories, by comparing the F1 scores of PG and MRL-CQA, it could be observed that NSM performed competitively. Furthermore, NSM performed the second best in "Simple Question" and "Comparative Reasoning" categories. This helps to validate the effectiveness of our proposed techniques. However, NSM is worse than MRL-CQA in six out of the seven categories. This verifies the adaptability of our model, which can quickly adapt to new tasks by employing the learned task-specific knowledge.

Note that our model was trained only on 1% of the training set, whereas the baseline models use the entire training set. Besides, our method trains one model to solve all questions, while CIP-Sep trains seven models, one for each category of problems. Thus our model is compared with seven individually trained models in CIPITR but still achieves the best overall performance, demonstrating the effectiveness of our technique.

3.2 Model Analysis

We conduct an ablation experiment to study the effect of meta-learning. We also study the effect of smaller training samples by comparing MRL-CQA's performance trained on 500 and 1K samples, against 2K used in the full model.

Table 2: Ablation study on the test set on macro F1 score change with different sizes of training samples.

Feature	Overall macro F1
PG	75.40%
MRL-CQA	
500-training	+0.01%
1,000-training	+0.58%
2,000-training	+2.31%

Table 2 summarizes the ablation study result. Trained on 500 samples only, MRL-CQA slightly improves performance by 0.01 percentage points compared to PG. When training sample increases to 1K, MRL-CQA outperforms PG by 0.58 percentage points. The full MRL-CQA model, trained on 2K samples, achieves a performance improvement over PG of 2.31 percentage points. These results demonstrate the ability to design a specific model for answering each question precisely, which is afforded by meta-learning.

3.3 Case Study

We provide a case study of different types of questions that MRL-CQA could answer, but our RL-based model, aka PG, could not solve. The comparison is given in Table 3, which lists the action sequences and the corresponding results these two models predicted when answering the same questions. We highlight the different parts of the action sequences that the two models generated.

For example, when answering the Logical Reasoning question in Table 3, PG was confused about what relations should be used to form feasible actions. It could be seen that PG failed to distinguish the two different relations for the two actions and thus produced a wrong answer.

Similarly, when answering the Verification question in Table 3, PG also yielded an infeasible action sequence. After forming a set of political territories that Hermine Mospointner is a citizen of, the bool action should be used to judge whether Valdeobispo and Austria are within the set. It can be seen that PG missed one action: *Bool (Austria)*.

The different optimization goals lead to the different results of the two models. PG, as a typical one-size-fits-all model, aims to estimate the globally optimal parameters by fitting itself to the training samples. Such a model extracts the information from the training data to update model parameters, applies the parameters to the new samples without modification thereafter. Therefore, when facing a wide variety of questions, it is hard for the model to find a set of parameters that fits all samples. Under the circumstances, like what is presented in Table 3, such a one-size-fits-all model could not handle the questions well.

However, our MRL-CQA model aims to learn general knowledge across tasks and fix the knowledge into the initial parameters. We thus learn a model that can subsequently adapt the initial parameters to each new given question and specialize the adapted parameters to the particular domain of the new questions. Therefore, with the help of the adapted parameters, MRL-CQA can answer each new question more precisely than PG.

4 Related Work

Imitation Learning. Imitation Learning aims to learn the policy based on the expert's demonstration by supervised learning. Saha et al. (2018) propose a CQA model that combines Hierarchical Recurrent Encoder-Decoder (HRED) with a Key-Value memory (KVmem) network and predicts the answer by attending on the stored memory. Guo et al. (2018) present a Dialog-to-Action (D2A) approach to answer complex questions by learning from the annotated programs. D2A employs a deterministic BFS procedure to label questions with pseudo-gold actions and trains an encoder-decoder model to generate programs by managing dialog memory. Multi-task Semantic Parsing (MaSP) (Shen et al., 2019) jointly optimizes two modules to solve the CQA task, i.e., entity linker and semantic parser, relying on annotations to demonstrate the desired behaviors. Different from the above approaches, our model performs better while removing the need for annotations.

Neural Program Induction (NPI). NPI is a paradigm for mapping questions into executable programs by employing neural models. Neural-Symbolic Machines (NSM) (Liang et al., 2017) is proposed to answer the multi-hop questions. NSM annotates the questions and then anchors the model to the high-reward programs by assigning them with a deterministic probability. Neural-Symbolic Complex Question Answering (NS-CQA) model (Hua et al., 2020) augments the NPI approach with a memory buffer to alleviate the sparse reward and data inefficiency problems appear in the CQA task. Complex Imperative

Table 3: A comparison of the action sequences and results that PG (the second column) and MRL-CQA (the third column) yield when answering the same questions.

An Example of Logical Reasoning Question		
Question Information	PG	MRL-CQA
Question: Which occupations are the professions of Sergio Piacentini **or** were a position held by Antoinette Sandbach?	**Action sequence:** *Select (Sergio Piacentini, position_held, occupation)* *Union (Antoinette Sandbach, position_held, occupation)*	**Action sequence:** *Select (Sergio Piacentini, occupation_of, occupation)* *Union (Antoinette Sandbach, position_held, occupation)*
Ground-truth answer: Member of the National Assembly for Wales, association football manager, association football player	**Execution result:** Member of the National Assembly for Wales	**Execution result:** Member of the National Assembly for Wales, association football manager, association football player
An Example of Verification (Boolean) Question		
Question Information	PG	MRL-CQA
Question: Is Hermine Mospointner a civilian of Valdeobispo **and** Austria?	**Action sequence:** *Select (Hermine Mospointner, country_of_citizenship, political territory)* *Bool (Valdeobispo)*	**Action sequence:** *Select (Hermine Mospointner, country_of_citizenship, political territory)* *Bool (Valdeobispo)* *Bool (Austria)*
Ground-truth answer: False and True	**Execution result:** False	**Execution result:** False and True

Program Induction from Terminal Rewards (CIPITR) (Saha et al., 2019) relies on auxiliary awards, KB schema, and inferred answer types for training an NPI model to solve the CQA task. However, CIPITR separately trains one model for each category of questions with a different difficulty level. Compared with the NPI models, our model can flexibly adapt to the question under processing.

Meta-learning. Meta-learning, aka learning-to-learn, aims to make learning a new task more effective based on the inductive biases that are meta-learned in learning similar tasks in the past. Huang et al. (2018) use MAML to learn a Seq2Seq model to convert questions in WikiSQL into SQL queries. More closely related to our work, Guo et al. (2019) propose Sequence-to-Action (S2A) by using MAML to solve CQA problems. They label all the examples in training set with pseudo-gold annotations, then train an encoder-decoder model to retrieve relevant samples and a Seq2Seq based semantic parser to generate actions based on the annotations. Unlike S2A, we introduce a Meta-RL approach, which uses RL to train an NPI model without annotating questions in advance.

5 Conclusion

CQA refers to answering complex natural language questions on a KB. In this paper, we propose a meta-learning method to NPI in CQA, which quickly adapts the programmer to unseen questions to tackle the potential distributional bias in questions. We take a meta-reinforcement learning approach to effectively adapt the meta-learned programmer to new questions based on the most similar questions retrieved. To effectively create the support sets, we propose an unsupervised retriever to find the questions that are structurally and semantically similar to the new questions from the training dataset. When evaluated on the large-scale complex question answering dataset, CQA, our proposed approach achieves state-of-the-art performance with overall macro and micro F1 score of 66.25% and 77.71%, respectively.

In the future, we plan to improve MRL-CQA by designing a retriever that could be optimized jointly with the programmer under the meta-learning paradigm, instead of manually pre-defining a static relevance function. Other potential directions of research could be toward learning to cluster questions into fine-grained groups and assign each group a set of specific initial parameters, making the model finetune the parameters more precisely.

Acknowledgments

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), Australian Research Council (DP190100006), the Judicial Big Data Research Centre, School of Law at Southeast University, and the project no. 31511120201 and 31510040201.

References

Ghulam Ahmed Ansari, Amrita Saha, Vishwajeet Kumar, Mohan Bhambhani, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019. Neural program induction for kbqa without gold programs or query annotations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4890–4896. AAAI Press.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2018. Dialog-to-action: conversational question answering over a large-scale knowledge base. In *Advances in Neural Information Processing Systems*, pages 2942–2951.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2019. Coupling retrieval and meta-learning for context-dependent semantic parsing. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 855–866. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. 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)*, volume 1, pages 1051–1062.

Yuncheng Hua, Yuan-Fang Li, Guilin Qi, Wei Wu, Jingyao Zhang, and Daiqing Qi. 2020. Less is more: Data-efficient complex question answering over knowledge bases. *Journal of Web Semantics*, Accepted.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 732–738.

Hailong Jin, Chengjiang Li, Jing Zhang, Lei Hou, Juanzi Li, and Peng Zhang. 2019. XLORE2: large-scale cross-lingual knowledge graph construction and application. *Data Intell.*, 1(1):77–98.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. 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)*, volume 1, pages 23–33.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Alex Nichol and John Schulman. 2018. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019. Complex program induction for querying knowledge bases in the absence of gold programs. *Transactions of the Association for Computational Linguistics*, 7:185–200.

Amrita Saha, Vardaan Pahuja, Mitesh M Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. 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*.

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

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57:78–85.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 643–648.

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

Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. 2016. Simple question answering by attentive convolutional neural network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1746–1756.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 571–581.