RLogic: Recursive Logical Rule Learning from Knowledge Graphs

Kewei Cheng
Department of Computer Science, University of California, Los Angeles, Los Angeles, CA
viviancheng@cs.ucla.edu

Jiahao Liu
Department of Computer Science, Brown University, Providence, RI
jiahao_liu@brown.edu

Wei Wang
Department of Computer Science, University of California, Los Angeles, Los Angeles, CA
weiwang@cs.ucla.edu

Yizhou Sun
Department of Computer Science, University of California, Los Angeles, Los Angeles, CA
yzsun@cs.ucla.edu

ABSTRACT

Logical rules are widely used to represent domain knowledge and hypothesis, which is fundamental to symbolic reasoning-based human intelligence. Very recently, it has been demonstrated that integrating logical rules into regular learning tasks can further enhance learning performance in a label-efficient manner. Many attempts have been made to learn logical rules automatically from knowledge graphs (KGs). However, a majority of existing methods entirely rely on observed rule instances to define the score function for rule evaluation and thus lack generalization ability and suffer from severe computational inefficiency. Instead of completely relying on rule instances for rule evaluation, RLogic defines a predicate representation learning-based scoring model, which is trained by sampled rule instances. In addition, RLogic incorporates one of the most significant properties of logical rules, the *deductive nature*, into rule learning, which is critical especially when a rule lacks supporting evidence. To push deductive reasoning deeper into rule learning, RLogic breaks a big sequential model into small atomic models in a *recursive* way. Extensive experiments have demonstrated that RLogic is superior to existing state-of-the-art algorithms in terms of both efficiency and effectiveness.

CCS CONCEPTS

• **Computing methodologies** → *Reasoning about belief and knowledge.*

KEYWORDS

Logical Rule Learning; Recursive; Knowledge Graph

ACM Reference Format:
Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. 2022. RLogic: Recursive Logical Rule Learning from Knowledge Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539421

1 INTRODUCTION

Although deep learning has achieved groundbreaking improvements in a variety of domains, e.g. speech recognition [7], image classification [15], and machine translation [2], the lack of interpretability and generalizability limits its application in broader tasks. Logical reasoning, which aims to leverage logical rules to derive missing knowledge, can provide insights into the inferred results and allow easy generalization to unobserved objects. To overcome the deficiency of deep learning, significant efforts have been made to integrate logical reasoning into neural network learning for various real-world applications, e.g., image understanding [1], visual question answering [18], and KG completion [27].

Due to the potential benefits brought by logical rules, increasing attention has been made to summarize the underlying patterns shared in the data to learn logical rules *automatically*. Note that logical rule is a schema level concept, while only instance level evidence can be directly observed from KGs. For example, from the KG in Fig. 1, we can observe closed paths (rule instances) such as:

$$\begin{aligned} \mathcal{CP}_1 &:= \text{Amy} \xrightarrow{\text{hasMother}} \text{Bess} \xrightarrow{\text{hasMother}} \text{Cara} \xleftarrow{\text{hasGrandma}} \text{Amy} \\ \mathcal{CP}_2 &:= \text{Dana} \xrightarrow{\text{hasMother}} \text{Eva} \xrightarrow{\text{hasMother}} \text{Faye} \xrightarrow{\text{hasSon}} \text{Gino} \xleftarrow{\text{hasUncle}} \text{Dana} \end{aligned} \tag{1}$$

To bridge the gap between instance level observation and schema level abstraction, the frequency of *rule instances* is widely utilized to define the plausibility of the logical rules. Traditional methods are represented by association rule mining [12, 19], where the score is defined as the ratio between the total number of rule instances and the total number of body instances for the corresponding rule. Then the rule mining process is to search over the rule space and select the rules with the highest score. For example, given the KG in Fig. 1, the rule space with length 2 to predict head relation r_2 (hasGrandma) can be represented as a tree structure, and each path from root to leaf corresponds to a rule body. Two rules with high scores can be found for Fig. 1 (δ_1 has a score of 0.75 and δ_2 has a score of 1), which are:

$$\begin{aligned} \delta_1 &:= \text{hasGrandma}(x,y) \leftarrow \text{hasMother}(x,z) \wedge \text{hasMother}(z,y) \\ \delta_2 &:= \text{hasUncle}(x,y) \leftarrow \text{hasMother}(x,z_1) \wedge \text{hasMother}(z_1,z_2) \wedge \text{hasSon}(z_2,y) \end{aligned} \tag{2}$$

Recently, neural network-based approaches are proposed [29, 42, 43], which enables rule instances to be softly counted via sequences of differentiable tensor multiplication. They also learn rules by searching for the rule bodies (or proof paths) that maximize the observations (i.e., triples) of the rule head.

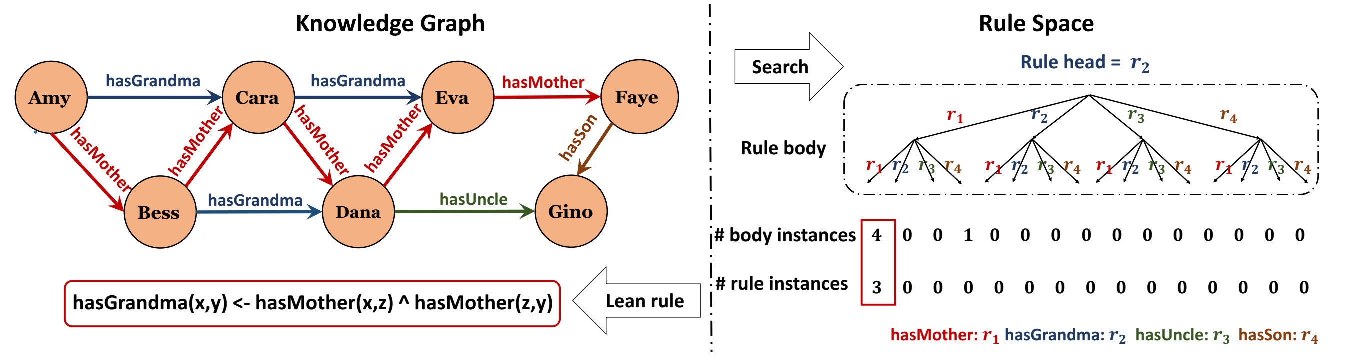


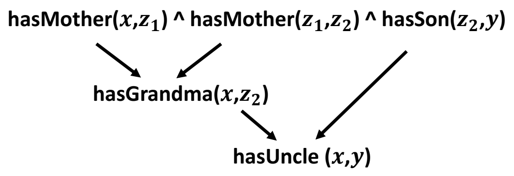
Figure 1: Left: An example KG and a detected rule with length 2. Right: A search tree for rules with length 2 and head relation r_2. A path from root to leaf corresponds to a rule body. Together with the head relation, it forms a candidate rule.

Although many efforts have been made to learn logical rules automatically, there are two limitations for the existing studies. First, most existing methods entirely rely on observed rule instances to define the score function for rule evaluation . They are unable to mine rules that have no support from rule instances. For example, due to the lack of support evidence, the following rule cannot be learned from Fig. 1:

$$\delta_3 := \text{hasUncle}(x, y) \leftarrow \text{hasGrandma}(x, z) \wedge \text{hasSon}(z, y) \quad (3)$$

Note that the number of rule instances is extremely large for a big KG, scalability is another central challenge. Instead of completely relying on rule instances for rule evaluation, we propose to learn logical rules directly in the schema level via representation learning-based model. The score of a rule can be calculated based on the learned predicate representations. In this way, a rule can be evaluated without the support from rule instances. Since a small amount of sampled rule instances are enough for training such model, it greatly improves the efficiency.

Second, a majority of existing methods learn logical rules by assuming that logical rules are independent from each other, which seriously contradict the *deductive nature* of logical rules. Deductive nature of logical rules describes the capability to combine existing rules for deriving new rules. For example, given two short rules δ_1 and δ_3, we can infer a long rule δ_2. The inference of δ_2 can be decomposed into recursive steps as shown in Fig. 2. Starting from the first two predicates, we can derive an intermediate conclusion $hasGrandma(x, z_2)$ following δ_1. Then, by replacing the first two predicates with the derived relation, we rewrite the rule body as $hasGrandma(x, z_2) \wedge hasMother(z_2, y)$ and derive the final conclusion according to δ_3. Because deductive nature describes the logical dependency among rules, it can be considered as "higher-order constraints" over rules, which is essential for us to validate rules without enough support from rule instances. For example, although we cannot rely on rule instances to evaluate δ_3 due to the lack of support evidence, as long as we can see evidence from Fig. 1 to support δ_1 and δ_2, by pushing deduction deeper into δ_2, δ_3 is forced to be true to make δ_2 true. To incorporate deductive nature into rule learning, we propose to break a big sequential model to small atomic models in a recursive way following logical deduction. The main contributions of this paper are summarized as follows:



hasMother(x,z_1) ^ hasMother(z_1,z_2) ^ hasSon(z_2,y)
hasGrandma(x,z_2)
hasUncle (x,y)

Figure 2: Deduction from δ_1 and δ_3 to δ_2.

- We investigate the problem of logical rule learning and propose a novel framework, known as RLogic, to learn logical rules directly at the schema level via representation learning-based model.
- RLogic breaks a big sequential model to small atomic models in a recursive way to push *deductive reasoning* deeper into rule learning, which is critical when a rule lacks supporting evidence.
- We experimentally demonstrate that RLogic is superior to existing SOTA algorithms in terms of both effectiveness and efficiency.

2 PRELIMINARIES AND PROBLEM DEFINITION

Horn Rule in the Language of Symbolic Logic. First-order logic (FOL) provides an important way of knowledge representation in AI [10, 32]. Horn rules, as a special case of FOL rules, are composed of a body of conjunctive predicates and a single head predicate. In this paper, we are interested in mining **chain-like** [1] Horn rules in the following form.

$$r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y) \quad (4)$$

where $r_h(x, y)$ is called **rule head** and $r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)$ is called **rule body**. Combining rule head and rule body, we denote a Horn rule as $(r_h, \mathbf{r_b})$ where $\mathbf{r_b} = [r_{b_1}, \ldots, r_{b_n}]$. The **length of a Horn rule** is defined as the number of predicates appearing in its body. In the area of symbolic logic, relations are called **predicates**. δ_1, δ_2 and δ_3 are real-world examples of Horn rules. By substituting the variables in a Horn rule with concrete entities, we get a **rule**

[1]The assumption of chain-like Horn rules prevents finding rules with unrelated relations.

Symbolic Logic	Knowledge Graph
Predicate	Relation
Rule Instance	Closed Path
Rule Head	Target Relation
Rule Body	Relation Path

Table 1: Comparison of different concepts in the language of symbolic logic v.s. knowledge graph.

instance. For example, a rule instance of δ_2 is given as follows:

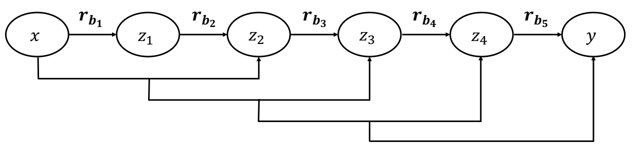
$$\text{hasUncle(Dana, Gino)} \leftarrow \text{hasMother(Dana, Eva)} \wedge \text{hasMother(Eva, Faye)} \wedge \text{hasSon(Faye, Gino)} \tag{5}$$

Horn Rule in the Language of Knowledge Graph. A KG, denoted by $\mathcal{G} = \{E, R, O\}$, consists of a set of entities E, a set of relations R and a set of observed facts O. Each fact in O is represented by a triple (e_i, r_k, e_j), where $e_i, e_j \in E$ and $r_k \in R$. Horn rule instances are called **closed paths** in the language of KG. For example, $\mathcal{CP}_2$ in Eq. (1) is the closed path corresponding to rule instance in Eq. (5). Note that logical rule is a schema level concept yet only instance level evidence in the form of closed paths can be directly observed from KGs. To bridge the gap between instance level observation and schema level abstract, we introduce **relation path** and **target relation** as follows. By ignoring all concrete entities along a closed path, we can divide a closed path into two components: (1) a relation path, which is defined as a sequence of relations $\mathbf{r_b} = [r_{b_1}, \ldots, r_{b_n}]$ through which two entities e_i and e_j can be connected on the graph. As shown in Fig. 1, [*hasMother*, *hasMother*, *hasSon*] is a relation path, through which *Dana* and *Gino* can be connected. A relation path corresponds to a conjunctive body of Horn rule in symbolic logic; and (2) a target relation, which is defined as a single relation r_t. It can close the relation path by connecting two entities e_i and e_j directly. In Figure 1, relation *hasUncle* is the target relation of relation path [*hasMother*, *hasMother*, *hasSon*]. A target relation corresponds to the head of a Horn rule in symbolic logic. Combining the relation path and target relation, we denote a closed path as $(r_t, \mathbf{r_b})$.

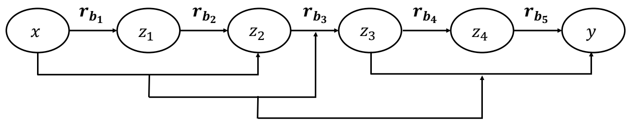
The Problem of Logical Rule Learning. Logical rule learning aims to assign a plausibility score $s(r_h, \mathbf{r_b})$ to each rule $(r_h, \mathbf{r_b})$ in **rule space**. $s(\cdot)$ is called the score function. The ratio that a body path can be closed is usually utilized to define the score function (e.g., the confidence for association rule mining [12] and the percentage of triples of the rule head to be satisfied for Neural-LP [42]). During rule extraction, top k rules with highest score will be selected as the learned rules.

3 APPROACH

In this section, we propose a novel framework - RLogic to learn rules in the *schema level*. Instead of completely relying on rule instances for rule evaluation, we propose to learn logical rules via *representation learning-based model*. As a small amount of sampled closed paths are enough for training such model, it greatly improves the efficiency. To push *deductive reasoning* deeper into rule learning, RLogic breaks a big sequential model to small atomic models in a *recursive* way, which is essential for us to detect rules without support from rule instances.



(a) Left-wise decomposition



(b) Decomposition following an irregular order

Figure 3: Different order to deduct a relation path. (a) left-wise decomposition; (b) irregular decomposition.

3.1 A New Measure for Rule Evaluation

Existing Measure for Rule Evaluation Among several possible measures, *confidence* is the most representative one widely used by association rule mining. It is defined as the ratio that a body path can be closed by the target relation. Given an arbitrary rule defined in Eq. 4, its confidence can be calculated as:

$$\frac{|\{(x, z_1, \ldots, z_{n-1}, y) : r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)\}|}{|\{(x, z_1, \ldots, z_{n-1}, y) : r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)\}|} \tag{6}$$

where the numerator is the number of its rule instances (i.e., closed paths) and the denominator is the number of its body instances (i.e., relation paths). Confidence cannot distinguish between "false statement" and "unknown statement". Therefore, it measures the rules purely based on the observed data and penalizes rules that make a large number of predictions in the unknown region, which make it easily affected by data bias.

Proposed Measure for Rule Evaluation The calculation of confidence entirely rely on observed rule instances. However, enumerating rule instances is usually time consuming. Instead, we propose a new measure for rule evaluation based on the probability that the rule body can be replaced by the rule head:

$$s(r_h, \mathbf{r_b}) = q(r_h = r_i | \mathbf{r_b}) \tag{7}$$

A sequential model, such as RNN, is a nature option to learn $q(r_h = r_i | \mathbf{r_b})$. We need a tensor with $|R|^{(l+1)}$ dimension to store the probabilities to learn rules with maximum length as l. It is too expensive and impossible to get enough data points to estimate each entry.

Note that deductive nature, as one of the most significant properties of logical rules, allows us to decompose the inference of long rules on the basis of the short rules. Given the short Horn rule that are in the form $r_h \leftarrow r_i \wedge r_j$, we can reduce the long relation path $r_{b_1}, r_{b_2} \ldots, r_{b_n}$ by replacing the relation pair $r_i \wedge r_j$ with their head r_h. By recursively applying different short Horn rules to a relation path, it will be transformed into a single head at the end. Following the same idea, we can use $q(r_h | r_i, r_j)$ to compute $q(r_h | \mathbf{r_b})$ in a recursive way. For example, given a relation path $[r_{b_1}, r_{b_2}, r_{b_3}]$, $q(r_h | r_{b_1}, r_{b_2}, r_{b_3})$ can be computed as follows if we follow the order from left to right to reduce the path:

$$q(r_h|r_{b_1}, r_{b_2}, r_{b_3}) = \sum_k q(r_h|r_k, r_{b_3})q(r_k|r_{b_1}, r_{b_2}) \tag{8}$$

As every step we only need to model a sequence with length 2, this significantly reduces the computational burden caused by long sequence modeling, such as RNN.

Although we can follow logical deduction to reduce a relation path into one single head, this head relation may not always be observed due to the sparsity of real-world KGs. To avoid penalizing rules that make predictions in the unknown region, we introduce $p(r_t|r_h)$ to bridge the gap between "ideal prediction" following logical rules and "real observation" given in KGs.

$$p(r_t|\mathbf{r_b}) = \sum_h p(r_t|r_h)q(r_h|\mathbf{r_b}) \tag{9}$$

where $p(r_t|\mathbf{r_b})$ is the ratio that a path $\mathbf{r_b}$ is closed.

3.2 Framework - RLogic

Following the proposed measurement, we introduce a *relation path encoder* and a *close ratio predictor* (i.e., predicting the ratio that a path will close) to model $q(r_h|\mathbf{r_b})$ and $p(r_t|r_h)$ respectively. Given a path $\mathbf{r_b}$, the relation path encoder first reduces $\mathbf{r_b}$ into a single head r_h by recursively applying different short Horn rules. Then the close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting the ratio that the path $\mathbf{r_b}$ is closed.

3.2.1 Predicate Representation Learning. Predicates are the basic units in context of logical rules. Low-dimensional vectors are learned to represent predicates. We denote the embedding of predicate r_i as $\mathbf{r_i} \in \mathbb{R}^d$. With the representation of predicates, RLogic can identify similar predicates like *GrandPa* and *GrandFather* even though they are lexically different. Note that we may not always find an existing relation to replace a relation path. For example, the relation path [*hasMother*, *hasMother*, *hasDaughter*], can be replaced by a single relation *hasAunt*, which may not be included in the KG. To accommodate unseen relations, we introduce a "null" predicate into the prediction set R, which is denoted as R_0. The score function is computed based on the representation of predicates. For any candidate rule, even without direct evidence support from its rule instances, we can still compute its score.

3.2.2 Relation Path Encoder. The goal of relation path encoder is to find a head relation r_h to replace a relation path $\mathbf{r_b}$. It requires learning both the order to deduct a relation path (i.e. the particular relation pair to be selected in each step) and the probability to replace a relation pair with a single relation (i.e. $q(r_h|r_i, r_j)$).

Learning the order to deduct a relation path. As shown in Figure 3, there are many different ways to deduct a relation path. Determining the best order involves search over a potentially large problem space. Given a relation path with length l, there are $C_{l-1} = \prod_{k=2}^{l-1} \frac{l-1+k}{k}$ (Catalan number) different ways to decompose the relation path. To alleviate the high computational complexity, instead of enumerating all possible deduction orders to find the global optimal, we adopt greedy algorithm to select the optimal relation pair at each step, which reduces the complexity into $(l-1)+(l-2)+\cdots+1$.

Considering that the ground truth deduction order is unavailable in our problem, we have to exploit different criteria to evaluate a relation pair in a path. Entropy is commonly used to measure confidence of predictions in the semi-supervised learning and domain adaptation. By minimizing entropy, the confidence of predictions can be encouraged. The entropy of a relation pair (r_i, r_j) is defined as follows:

$$E((r_i, r_j)) = \sum_{r_h \in R} -q(r_h|r_i, r_j) \log q(r_h|r_i, r_j) \tag{10}$$

The lower the entropy, the more confident we are to find a relation to replace the relation pair. To maximize the confidence of deduction, we select the relation pair with the lowest entropy as the optimal every step.

Learning the probability to replace a relation pair with a single relation. Given the order to deduct a relation path, the next step is to learn $q(r_h|r_i, r_j)$ to reduce the relation path recursively. $q(r_h = r_k|r_i, r_j)$ can be approximated via an multilayer perceptron (MLP) classifier. The MLP classifier $f_\theta(\mathbf{r_i}, \mathbf{r_j})$ is a two-layer fully-connected neural network with parameters θ. It takes the embedding of predicate r_i, r_j as input, and outputs the probability that they can be replaced by each of the relation in KG plus the "null" predicate (i.e., unknown relation). Considering that $q(r_h|r_i, r_j)$ follows categorical distribution, the MLP classifier uses softmax as the activation function for the final layer. Using the function $f_\theta(\mathbf{r_k}, \mathbf{r_{b_3}})$ to approximate $q(r_h|r_k, r_3)$, we rewrite formula in Eq. (8) as:

$$q(r_h|r_{b_1}, r_{b_2}, r_{b_3}) = \sum_k q(r_k|r_{b_1}, r_{b_2}) f_\theta(\mathbf{r_k}, \mathbf{r_{b_3}}) \tag{11}$$

As can be observed from the formula, the relation pair r_{b_1}, r_{b_2} can be replaced by each of the relation in KG with different probability (i.e., $q(r_k|r_{b_1}, r_{b_2})$). It is significant to include all these probabilities into computation to model complex logical deduction. For example, given the rule body *hasAunt*$(x, z) \wedge$ *hasSister*(z, y), both *hasMother*(x, y) and *hasAunt*(x, y) can be derived as the rule head. However, this causes great computational burden as $f_\theta(\mathbf{r_k}, \mathbf{r_{b_3}})$ also need to be calculated for each of predicates r_k in a KG (i.e., $|R| + 1$ in total). To reduce the computational burden, we propose to approximate $\sum_k q(r_k|r_{b_1}, r_{b_2}) f_\theta(\mathbf{r_k}, \mathbf{r_{b_3}})$ with $f_\theta(\tilde{\mathbf{r}}, \mathbf{r_{b_3}})$ instead, where $\tilde{\mathbf{r}}$ can be defined as:

$$\tilde{\mathbf{r}} = \sum_k q(r_k|r_{b_1}, r_{b_2}) \cdot \mathbf{r_k} \tag{12}$$

We can observe that $\tilde{\mathbf{r}}$ is a "weighted average" representation, which is learned by "softly" adding up representations of all predicates r_k in a KG. With $\tilde{\mathbf{r}}$, we significantly reduce the computational burden.

3.2.3 Close Ratio Predictor. Although we can follow logical deduction to reduce a relation path into one single head relation, this head relation may not always be observed due to the sparsity of real-world KGs. To bridge the gap between "ideal prediction" following logical rules and "real observation", close ratio predictor is proposed to predict the ratio that a path will close. A two-layer fully-connected neural network (MLP) is introduced to model $p(r_t|r_h)$. In particular, it uses ReLU as the activation function for the first layer and adds sigmoid as the activation function for the second layer. The "weighted average" $\tilde{\mathbf{r}}$ learned by the relation path encoder at the final step and the embedding $\mathbf{r_t}$ are taken as input to decide the close ratio.

3.2.4 Model Training. This section discusses training procedures for RLogic. We first introduce our proposed closed path sampler for training data generation. Then we give the objective functions for training relation path encoder and close ratio predictor.

Closed Path Sampler for Training Data Generation. Rather than enumerate all closed paths in KG, we utilize a closed path sampler to sample only a small portion of closed paths to train the model. We propose a random walk [33] based procedure to efficiently sample the close paths. Formally, given a source entities x_0, we simulate a random walk of fixed length n. Let x_i denote the ith node in the walk. Nodes x_i are generated by the following distribution:

$$p(x_i = e_i | x_{i-1} = e_j) = \begin{cases} \frac{1}{|\mathcal{N}(e_j)|}, & \text{if } (e_i, e_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where $|\mathcal{N}(e_j)|$ is the neighbor size of entity e_j. Different from random walk, each time after we sample the next node x_i, we add all edges which can directly connect x_0 and x_i in KG to construct the closed paths.

Objective function for training relation path encoder. Relation path encoder aims to find a head relation r_h to replace a relation path $\mathbf{r_b}$. Each sampled closed path can be regarded as a positive example, whose target relation gives the ground truth of head relation r_h. Negative examples can be generated by corrupting the positive examples. Note that KGs operate under the Open World Assumption (OWA). A statement that is not contained in the KG is not necessarily false. It is just unknown. Rather than assuming negative examples are necessarily false, we can only infer that they are more invalid than those positive ones. To make the scores of positive examples higher than those of negative ones, we employ pairwise margin-based ranking loss function to learn $q(r_k|\mathbf{r_b})$.

$$\sum_{(r_h,\mathbf{r_b})\in\mathcal{P}} \sum_{(r_h{}',\mathbf{r_b})\in\mathcal{N}(r_h,\mathbf{r_b})} [q(r_h|\mathbf{r_b}) + \gamma - q(r_h{}'|\mathbf{r_b}))]_+ \tag{14}$$

where $[x]_+ = \max\{0, x\}$ and the scoring function $s(\cdot)$ can be parameterized by the relation path encoder. $\gamma > 0$ is the margin hyperparameter separating positive and negative closed paths. To reduce the effects of randomness, we sample multiple negative examples for each positive closed path. We denote the set of negative samples of a closed path $(r_h, \mathbf{r_b})$ as $\mathcal{N}(r_h, \mathbf{r_b})$, which is constructed by replacing the head relation of the closed path $(r_h, \mathbf{r_b})$ with a relation sampled randomly from relation set R:

$$\mathcal{N}(r_h, \mathbf{r_b}) \subset \{(r_h{}', \mathbf{r_b}) | r_h{}' \in R\}\} \tag{15}$$

Objective function for training close ratio predictor. Close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting the ratio that a path $\mathbf{r_b}$ will close. Each closed path in KG can be regarded as a positive example while the paths fail to be closed by any relation are regarded as negative examples. The training objective for learning $p(r_t|r_h)$ can be formulated as the binary cross-entropy loss:

$$\sum_{(r_t,r_h)\in\mathcal{P}} \log p(r_t|r_h) + \sum_{(r_t,r_h)\in\mathcal{N}} \log(1 - (p(r_t|r_h)) \tag{16}$$

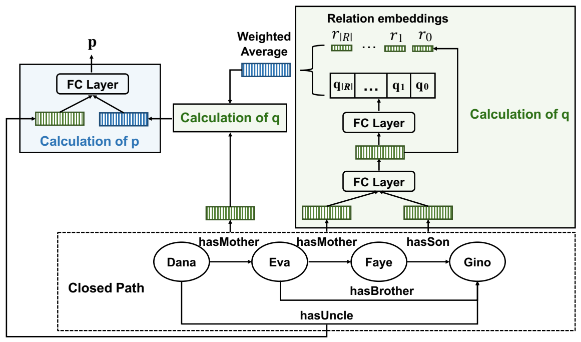


Figure 4: Illustration of the RLogic framework. It contains two main components: (1) a relation path encoder to model $q(r_h|\mathbf{r_b})$; and (2) a close ratio predictor to model $p(r_t|r_h)$. Given a path $\mathbf{r_b}$, the relation path encoder reduces $\mathbf{r_b}$ into a single head r_h by recursively merge relation pairs in path $\mathbf{r_b}$ according to $q(r_h|r_i, r_j)$. Then, the close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting predicting the ratio that the relation path $\mathbf{r_b}$ will close.

where $\mathcal{P}$ stores only positive examples and $\mathcal{N}$ stores only negative examples. r_h is the head to replace the relation path $\mathbf{r_b}$ learned by the relation path encoder.

3.3 Rule Extraction

To recover logical rules from RLogic, we calculate the score $s(r_h, \mathbf{r_b})$ for each rule δ in rule space when the model has finished training. Unlike most of the existing methods, which can only rank rules conditioned on head relation, our method can rank rules globally. With such scores, we can select the most important rules to interpret the whole KG rather than to infer a certain relation. The detailed procedure to extract rules is presented as below. Given a candidate rule $(r_h, \mathbf{r_b})$, we reduce the rule body $\mathbf{r_b}$ into a single head r_h by recursively merge relation pairs in path $\mathbf{r_b}$ according to $q(r_h|r_i, r_j)$. Every step, the relation pair (r_i, r_j) with minimum entropy will be selected and replaced by an intermediate representation $\tilde{\mathbf{r}}$ according to Eq. (12). At the end of the deduction, we obtain the vector $[q(r_0|\mathbf{r_b}), q(r_1|\mathbf{r_b}), \ldots, q(r_{|R|}|\mathbf{r_b})]$ where $q(r_k|\mathbf{r_b})$ are the score of rule $(r_k, \mathbf{r_b})$. Top k rules with highest score will be selected as learned rules.

4 EXPERIMENTS

Datasets. Four widely used benchmark datasets are adopted to evaluate our proposed RLogic, which includes WN18RR [8], FB15K-237 [37], YAGO3-10 [34] and Family [13]. The detailed statistics of the datasets are summarized in Appendix.

4.1 Quality of Learned Rules in Terms of KG Completion Task

KG completion is a classic task widely used by logical rule learning methods such as Neural-LP [42], DRUM [29] and NLIL [43] to evaluate the quality of learned rules. An existing algorithm called *forward chaining* [30] can used to derive missing facts from logical rules efficiently. We selected the top 2400 rules with the highest score

Category	Model	WN18RR			FB15K-237			YAGO3-10		
		MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10
KGE	TransE	0.23	2.2	52.4	0.29	18.9	46.5	0.36	25.1	58.0
	DistMult	0.42	38.2	50.7	0.22	13.6	38.8	0.34	24.3	53.3
	ConvE	0.43	40.1	52.5	**0.32**	21.6	50.1	0.36	26.5	55.6
	ComplEx	0.44	41.0	51.2	0.24	15.8	42.8	0.34	24.8	54.9
	RotatE	**0.47**	42.9	**55.7**	**0.32**	**22.8**	**52.1**	**0.49**	**40.2**	**67.0**
Rule Learning	Neural-LP†	0.38	36.8	40.8	0.24	17.3	36.2	-	-	-
	NLIL†	0.30	20.1	33.5	0.25	13.8	32.4	-	-	-
	DRUM†	0.38	36.9	41.0	0.23	17.4	36.4	-	-	-
	AMIE	0.36	39.1	48.5	0.23	14.8	41.9	0.25	20.6	34.3
	RNNLogic (w/o emb)‡	0.46	41.4	53.1	0.29	20.8	44.5	-	-	-
	RLogic	**0.47**	**44.3**	53.7	0.31	20.3	50.1	0.36	25.2	50.4

† Neural-LP, NLIL and DRUM exceeds the capacity of our machines on YAGO3-10 dataset

‡ Results on RLogic are taken from the original papers.

Table 2: Transductive link prediction. The bold numbers represent the best performances among all methods while the underlined numbers represent the best performances among all rule learning methods.

learned by RLogic for the KG completion task. Detailed method of applying rules for KG completion is available in Appendix.

Evaluation Metrics. We mask the head or tail entity of each test triple, and require each method to predict the masked entity. During evaluation, we use the filtered setting [3] and three evaluation metrics, i.e., Hit@1, Hit@10 and MRR. To break ties for triples with the same score, we follow *random protocol* [36] to rank the triples with the same score.

Comparing with Other Methods. We evaluate RLogic against SOTA algorithms, including: (1) traditional KG embedding (KGE) methods (e.g., TransE [3], DistMult [41], ConvE [8], ComplEx [38] and RotatE [35]); (2) logical rule learning methods (e.g., Neural-LP [42], NLIL [43], DRUM [29], AMIE [12] and RNNLogic [26]). More detailed settings are in Appendix. The comparison results are presented in Table 2. We can observe that: (1) Although RLogic is not designed specifically for KG completion task, compared with traditional KGE models, it still achieves comparable result on all datasets; (2) RLogic outperforms most logical rule learning methods with significant performance gain in most cases; (3) RNNLogic shows great performance over KG completion tasks because it jointly trains a powerful reasoning predictor to predict missing links. To fairly compare with RNNLogic, we propose RLogic+ to incorporate an improved reasoning predictor for prediction. We will show the results of RLogic+ on KG completion task in the following section.

Inductive Link Prediction. It is important to note that comparing logical rule learning methods with KGE methods solely on transductive KG completion task is not fair. Different from KGE methods, which are not capable of reasoning on unseen entities, logical rules are more powerful in inductive setting. The results for the inductive link prediction experiment are shown in Table 3. More detailed settings are in Appendix. We observe that all rule learning algorithms still achieve similar performances as they did in transductive setting.

RLogic+. Unlike other baseline methods, RLogic directly learns rules without enhancing KG completion tasks as a side product. Although RLogic is able to generate high-quality logic rules, its performance over KG completion task is severely limited by the coverage of rules and the incompleteness of KGs due to the lack of a good reasoning predictor. Following UniKER [4], we resolve the KG sparsity issue by adding additional triples with high score using RotatE [35], and then conduct forward chaining to predict missing triples. The results are shown in Table 4. We can see that under the help of KG embedding, the performance of RLogic+ over KG completion task gets significantly improved on all datasets, especially on FB15k237.

4.2 Quality of Learned Rules in Terms of Rule Head Prediction Task.

In addition to the promising performance in term of KG completion task, *we also directly evaluate the correctness of rules learned by each system on Family dataset.* In particular, we propose a novel task **rule head prediction** to predict the heads of a set of rule bodies. *Human annotation are used to label the rule bodies for ground truth preparation.* Mean Average Precision (MAP) are taken as evaluation metric. More detailed settings are in Appendix.

4.2.1 Learn Equal-length Rules. We evaluate RLogic against a number of logical rule learning methods. Each method is given a set of closed paths with length 2 as training data and required to predict the rule head of *equal-length* rule bodies. The comparison results are presented in Figure 5. We observed that RLogic outperforms all other logical rule learning methods with significant gain.

4.2.2 Learn Longer Rules. RLogic incorporates the deductive nature into rule learning and thus can learn longer rules with only shorter closed paths observed in the training stage. In this experiment, we allow only closed paths with length 2 being observed in the training stage and require each system to predict rule heads of *longer rule bodies* whose lengths range from 2 to 6.

Comparing with Other Methods. Considering that most of existing logical rule learning methods rely on rule instances to define the score function for rule evaluation and thus cannot handle such difficult setting, we construct two baselines by replace the relation path encoder in RLogic with RNN and LSTM. Experiment results are shown in the left figures of Figure 6. We observe that: (1) RLogic performs the best in rule head prediction task, giving almost completely correct predictions; (2) LSTM performs better

Model	WN18RR			FB15K-237			YAGO3-10		
	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10
KGE[†]	-	-	-	-	-	-	-	-	-
Neural-LP[‡]	0.23	20.3	33.1	0.14	9.3	27.6	-	-	-
DRUM[‡]	0.23	20.5	34.4	0.16	10.8	29.3	-	-	-
AMIE	0.32	33.6	45.5	0.19	13.9	38.0	0.21	15.8	30.1
RLogic	**0.43**	**42.1**	**50.8**	**0.29**	**18.4**	**48.7**	**0.32**	**22.8**	**47.2**

[†] KGE methods are not applicable in inductive setting.
[‡] Neural-LP, NLIL and DRUM exceeds the capacity of our machines on YAGO3-10 dataset
[*] Results on RLogic are taken from the original papers.

Table 3: Inductive link prediction. The bold numbers represent the best performances among all methods.

Model	WN18RR			FB15K-237			YAGO3-10		
	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10
RNNLogic+ (with emb.)[†]	0.51	**47.1**	59.7	0.35	25.8	53.3	-	-	-
RLogic+	**0.52**	46.6	**60.4**	**0.55**	**51.1**	**64.3**	**0.53**	**42.6**	**70.3**

[†] Results on RLogic are taken from the original papers.

Table 4: Combine learned rules with KG embedding for KG completion task.

Methods	WN18-RR	FB15K-237	YAGO3-10
Neural-LP[†]	21.8	395.0	-
NLIL[†]	14.9	108.3	-
DRUM[†]	19.1	373.8	-
AMIE	0.5	13.9	41.3
RNNLogic[†]	17.4	-	> 4 days
RLogic	**0.2**	**5.2**	**17.3**

[†] Neural-LP, NLIL and DRUM exceeds the capacity of our machines on YAGO3-10 dataset and RNNLogic exceeds the capacity of our machines on FB15K-237 dataset.

Table 5: Training time (minutes) of logical rule learning methods on WN18-RR, FB15K-237 and YAGO3-10 datasets.

than RNN, as it naturally can handle longer sequential data better compared with RNN.

Effect of Rule Length. To further investigate how the length of rule body affects the performance, we vary the lengths of rule bodies among $\{2, 3, 4, 5, 6\}$ and report corresponding MAP. As depicted in the right figure of Figure 4, the performances of RNN and LSTM drop severely when the rule bodies grow longer while RLogic is less affected by the length of rule body.

Case Study. In addition, we conduct a case study to show the power of the recursive mechanism which enables RLogic to give right predictions when rule body grows longer. As shown in the figure on the top of Figure 7, three queries are manually designed with rule bodies of length 2, 5 and 6 respectively. The longer queries are formed by adding new predicates to the end of rule body of the shorter ones. Top three rule heads inferred are given in the table at the bottom of Figure 7, associated with their probabilities (rounded). We observed that RLogic consistently performs well in all cases while it is more and more challenging for RNN and LSTM to provide correct prediction with the increase of body length.

4.3 Training Efficiency

RLogic learns rules directly at schema level while other existing methods learn rules based on instance-level ground rules. Therefore, RLogic is much more efficient than all existing methods. To demonstrate the scalability of RLogic, we give the training time of RLogic and other logical rule learning methods on three benchmark datasets in Table 5. Considering that it is challenging for baseline methods to learn long rules, to fairly compare with different methods, we limit the maximum length of learned rules to be 2. We can observe that: (1) Neural-LP, NLIL and DRUM do not perform well in terms of efficiency as they involve large matrix multiplication. They cannot handle YAGO3-10 dataset due to the memory issue; (2) It is also challenging for RNNLogic to scale to large scale KGs. It can neither handle KG with hundreds of relations (e.g., FB15K-237) nor KG with million entities (e.g., YAGO3-10); (3) Although performances of RLogic and AMIE are on the same scale, AMIE is less efficient as it relies on all rule instances for rule evaluation.

[2]The code provided by RNNLogic didn't output the learned rules. Without the learned rules, we are unable to include RNNLogic in this experiments.

4.4 Quality and Interpretability of the Rules

To demonstrate the quality and interpretability of rules mined by RLogic, we show some logic rules generated on the FB15k-237 dataset in Table 8. Two rules with different lengths are presented for each head predicate. We highlight the predicates which convey the same semantic meaning with boldface. We can observed that the boldfaced predicates in longer rules can be used to infer the boldfaced predicate in shorter rules. This observation again validates that RLogic is able to captures the deductive nature of logical rules. More analysis over the generated rules are in Appendix.

5 RELATED WORK

Association Rule Mining. Association rule mining learns rules by searching over a large rule space. The frequency of rule instances is used to estimate the plausibility of a specific logic. Traditional methods such as AMIE [11, 12], WARMR [6] and RLvLR [24] rely on all rule instances for rule evaluation. More recently, several attempts such as AnyBURL [19] and RARL [25] have been made to sample rule instances instead to accelerate rule learning. Since methods in this category rely on observed rule instances to define the score function for rule evaluation, they usually lack generalization ability (i.e., a rule cannot be detected without seeing a rule instance). Instead, RLogic defines a predicate representation learning-based

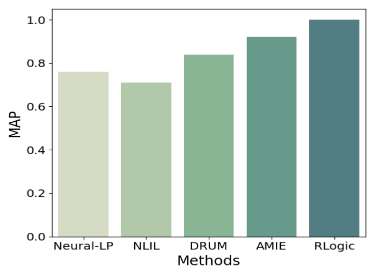


Figure 5: Equal-length rule detection comparison on Family dataset[2]

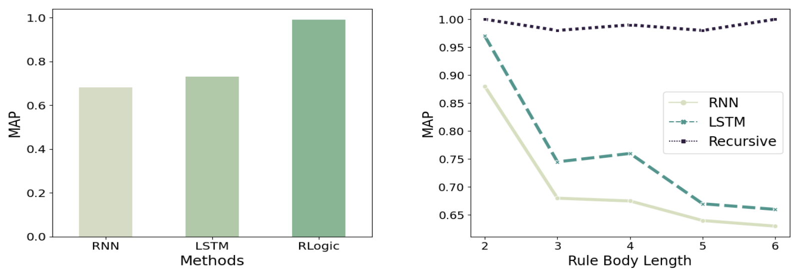


Figure 6: Longer-length rule detection comparison on Family dataset. Left: overall rule detection performance, where the lengths of rule bodies are varied among $\{2, 3, 4, 5, 6\}$. Right: rule detection performance w.r.t body length.

speak_language(x, y) ← geographic_distribution(x, z) ∧ **phone_service_language**(z, y)
speak_language(x, y) ← geographic_distribution(x, z_1) ∧ **tv_network_programs**(z_1, z_2) ∧ **program_language**(z_2, y)
location_at_time_zones(x, y) ← county_at_location(x, z) ∧ **location_at_time_zones**(z, y)
location_at_time_zones(x, y) ← county_at_location(x, z_1) ∧ **location_partially_contains**(z_1, z_2) ∧ **location_at_time_zones**(z_2, y)
has_nationality(x, y) ← write_tv_programs(x, z) ∧ **tv_programs_in_country**(z, y)
has_nationality(x, y) ← write_tv_programs(x, z_1) ∧ **has_regular_tv_appearance**(z_1, z_2) ∧ **headquarters_in_country**(z_2, y)

Table 6: Top rules learned by RLogic on FB15K-237. We highlight the predicates which convey the same semantic meaning with boldface.

Query 1 $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,y)$

Query 2 $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,z_2) \wedge \boldsymbol{hasFather}(z_2,z_3) \wedge \boldsymbol{hasWife}(z_3,z_4) \wedge \boldsymbol{hasHusband}(z_4,y)$

Query 3 $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,z_2) \wedge \boldsymbol{hasFather}(z_2,z_3) \wedge \boldsymbol{hasWife}(z_3,z_4) \wedge \boldsymbol{hasHusband}(z_4,z_5) \wedge \boldsymbol{hasBrother}(z_5,y)$

Query	RNN	LSTM	RLogic
Query 1	**hasSister 0.7**	**hasSister 0.9**	**hasSister 0.9**
	hasBrother 0.1	hasBrother 0.0	hasDaughter 0.0
	hasMother 0.1	hasNiece 0.0	hasBrother 0.0
Query 2	hasSister 0.8	**hasFather 0.8**	**hasFather 1.0**
	hasFather 0.1	hasMother 0.1	hasSon 0.0
	hasUncle 0.0	hasDaughter 0.1	hasMother 0.0
Query 3	hasBrother 0.3	hasMother 0.8	**hasUncle 0.8**
	hasDaughter 0.3	hasBrother 0.1	hasBrother 0.1
	hasSon 0.2	hasFather 0.1	hasAunt 0.0

Figure 7: Case study of RLogic in inferring rules with different lengths. The figure on the top gives rule bodies with different lengths (i.e., 2, 5, 6). The table at the bottom presents predicted rule heads ranked by probabilities. The bold predicates corresponds to the ground truth answers.

scoring model. For any candidate rule, even without direct evidence support from its rule instances, we can still compute its score.

Neural Logic Programming. Very recently, neural logic programming approaches are proposed to extend the rule mining problem from counting to learning. Methods such as Neural-LP [42] enables rule instances to be softly counted via sequences of differentiable tensor multiplication. An neural controller system based on attention mechanism is used to learn the score of a specific logic. However, Neural-LP could learn higher score of a meaningless rule because it shares an atom with a useful rule. To address this problem, RNNs are utilized in DRUM [29] to prune the potential incorrect rule bodies. In addition, Neural-LP can learn only chain-like Horn rules while NLIL [43] extends Neural-LP to learn Horn rules in more general form. Because neural logic programming approaches involve large matrix multiplication and simultaneously learn logic rules and their weights, which is nontrivial in terms of optimization, they cannot handle large KGs, such as YAGO3-10. To address this issue, RNNLogic [26]) is proposed to separate rule generation and rule weight learning by introducing a rule generator and a reasoning predictor respectively. Although the introduction of the rule generator reduces the search space, it is still challenging for RNNLogic to scale to knowledge graph with hundreds of relations (e.g., FB15K-237) or millions of entities (e.g., YAGO3-10).

Inductive Logic Programming. Mining Horn clauses has been extensively studied in the Inductive Logic Programming (ILP) [20, 21, 23]. Given a set of positive examples, as well as a set of negative examples, an ILP system aims to learn logic rules which are able to entail all the positive examples while exclude any of the negative examples. Some representative methods includes FOIL [28] and GOLEM [22] and ∂ILP [9]. Although ILP has shown its power in many areas [39, 44], scalability is a central challenge for ILP problem as it involves several steps that are NP-hard.

Reinforcement Learning Based Approaches. Most reinforcement learning based approaches aim to learn instance level reasoning paths in KGs to predict missing triples [5, 31, 40]. Logical rules are extracted from these instance level reasoning paths as the side product. Since the reward signals are usually sparse in KGs, training reinforcement learning agents is usually challenging [16]. Different from the above works, R5 [17] is recently proposed to directly learn logical rules based on reinforcement learning. Similar as our work, it also performs recurrent relational prediction. However, their work assumes that only one head can be derived from a relation pair and thus fails to model underlying complex logical deduction. Because the authors of R5 doesn't release the code and they conduct experiments over different datasets, we didn't include it as our baseline in the experiments.

6 CONCLUSION

A majority of existing methods entirely rely on observed rule instances to define the score function for rule evaluation and thus lack generalization ability and suffer from severe computational inefficiency. Instead of completely relying on rule instances for rule evaluation, RLogic defines a predicate representation learning-based scoring model, which is trained by sampled rule instances. In addition, RLogic incorporates one of the most significant properties of logical rules, the deductive nature, into rule learning, which is critical especially when a rule lacks supporting evidence. To push deductive reasoning deeper into rule learning, RLogic breaks a big sequential model into small atomic models in a recursive way.

ACKNOWLEDGEMENTS

This work was partially supported by NSF 1705169, 1937599, 2031187, 2106859, 2119643, NIH R35-HL135772, NIH/NIBIB R01-EB027650, Amazon Research Awards, Cisco research grant USA000EP280889, NEC Gifts, Picsart Gifts, and Snapchat Gifts.

REFERENCES

[1] Somak Aditya, Yezhou Yang, and Chitta Baral. 2019. Integrating knowledge and reasoning in image understanding. *arXiv preprint arXiv:1906.09954* (2019).
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. 2787–2795.
[4] Kewei Cheng, Ziqing Yang, Ming Zhang, and Yizhou Sun. [n. d.]. UniKER: A Unified Framework for Combining Embedding and Definite Horn Rule Reasoning for Knowledge Graph Inference. ([n. d.]).
[5] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851* (2017).
[6] Luc Dehaspe and Hannu Toivonen. 1999. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery* 3, 1 (1999), 7–36.
[7] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8599–8603.
[8] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[9] Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61 (2018), 1–64.
[10] Melvin Fitting. 2012. *First-order logic and automated theorem proving*. Springer Science & Business Media.
[11] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal* 24, 6 (2015), 707–730.
[12] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*. 413–422.
[13] Geoffrey E Hinton et al. 1986. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, Vol. 1. Amherst, MA, 12.
[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[16] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2018. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568* (2018).
[17] Shengyao Lu, Bang Liu, Keith G Mills, SHANGLING JUI, and Di Niu. 2022. R5: Rule Discovery with Reinforced and Recurrent Relational Reasoning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=2eXhNpHeW6E
[18] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584* (2019).
[19] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion.. In *IJCAI*. 3137–3143.
[20] Stephen Muggleton. 1992. *Inductive logic programming*. Number 38. Morgan Kaufmann.
[21] Stephen Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming* 19 (1994), 629–679.
[22] Stephen Muggleton, Cao Feng, et al. 1990. *Efficient induction of logic programs*. Citeseer.
[23] Shan-Hwei Nienhuys-Cheng and Ronald De Wolf. 1997. *Foundations of inductive logic programming*. Vol. 1228. Springer Science & Business Media.
[24] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. 2018. Scalable Rule Learning via Learning Representation.. In *IJCAI*. 2149–2155.
[25] Giuseppe Pirrò. 2020. Relatedness and TBox-Driven Rule Learning in Large Knowledge Bases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 2975–2982.
[26] Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. 2020. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. *arXiv preprint arXiv:2010.04029* (2020).
[27] Meng Qu and Jian Tang. 2019. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems*. 7710–7720.
[28] J. Ross Quinlan. 1990. Learning logical definitions from relations. *Machine learning* 5, 3 (1990), 239–266.
[29] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. 2019. Drum: End-to-end differentiable rule mining on knowledge graphs. *arXiv preprint arXiv:1911.00055* (2019).
[30] Eric Salvat and Marie-Laure Mugnier. 1996. Sound and complete forward and backward chainings of graph rules. In *International Conference on Conceptual Structures*. Springer, 248–262.
[31] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. 2018. M-walk: Learning to walk over graphs using monte carlo tree search. *arXiv preprint arXiv:1802.04394* (2018).
[32] Raymond M Smullyan. 1995. *First-order logic*. Courier Corporation.
[33] Frank Spitzer. 2013. *Principles of random walk*. Vol. 34. Springer Science & Business Media.
[34] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. 697–706.
[35] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).
[36] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. 2019. A re-evaluation of knowledge graph completion methods. *arXiv preprint arXiv:1911.03903* (2019).
[37] Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*. 57–66.
[38] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*. PMLR, 2071–2080.
[39] Kazuhisa Tsunoyama, Ata Amini, Michael JE Sternberg, and Stephen H Muggleton. 2008. Scaffold hopping in drug discovery using inductive logic programming. *Journal of chemical information and modeling* 48, 5 (2008), 949–957.
[40] Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690* (2017).
[41] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).
[42] Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*. 2319–2328.
[43] Yuan Yang and Le Song. 2019. Learn to Explain Efficiently via Neural Logic Inductive Learning. *arXiv preprint arXiv:1910.02481* (2019).
[44] John M Zelle and Raymond J Mooney. 1993. Learning semantic grammars with constructive inductive logic programming. In *AAAI*. 817–822.

Dataset	# Data	# Relation	# Entity
FB15K-237	310,116	237	14,541
WN18RR	93,003	11	40,943
YAGO3-10	1,089,040	37	123,182
Family	28,356	12	3007

Table 7: Data statistics.

A USE LOGICAL RULES FOR KNOWLEDGE GRAPH COMPLETION

An existing algorithm called **forward chaining** [30] can used to derive missing facts from logical rules efficiently.

Given a query $(h, r, ?)$, let A be the set of candidate answers which can be discovered by any learned rule using forward chaining. For each candidate answer $e \in A$, the score of triple (h, r, e) can be calculated as

$\sum_{\delta \in \text{rule space}} \sum_{\text{path} \in P(h,r,e)} s(\delta)$.

Taking the test KG in Figure. (1) in main context as an example. Starting from known facts (e.g., *hasMother (Lily, Jane)*, *hasMother (Jane, Betty)* and *hasSon (Betty, Tom)*), it triggers rule δ_2 whose premise is satisfied (e.g., *hasUncle (Lily, Tom)* $\leftarrow$ *hasMother (Lily, Jane)* $\wedge$ *hasMother (Jane, Betty)*) $\wedge$ *hasSon (Betty, Tom)*). The score of its conclusion (e.g., *hasUncle (Lily, Tom)*) becomes $s(\delta_2)$. Then, the conclusion *hasUncle (Lily, Tom)* will be added to the known facts as the inferred missing fact.

Considering that RLogic can only learn chain-like horn rules, which is in the form as shown in Eq. (5) in the main context. The conjunctive body of a ground chain-like Horn rules is essentially a path in a KG and thus can be extracted efficiently using sparse matrix multiplication.

B EXPERIMENT

B.1 Datasets

FB15K237, WN18RR and YAGO3-10 are the most widely used benchmark datasets for KGE models, which don't suffer from test triple leakage in the training set. The Family dataset is selected due to better interpretability and high intuitiveness. Because inverse relations are required to learn rules, we preprocess the KGs to add inverse links.

- FB15k-237 is the most commonly used benchmark knowledge graph datasets introduced in [37]. It is an online collection of structured data harvested from many sources, including individual, user-submitted wiki contributions.
- WN18RR is also a widely used benchmark knowledge graph datasets introduced in [8]. It is designed to produce an intuitively usable dictionary and thesaurus, and support automatic text analysis. Its entities correspond to word senses, and relationships define lexical relations between them.
- YAGO3-10 is another widely used benchmark knowledge graph datasets introduced in [34]. It is a subset of YAGO, which is a large semantic knowledge base, derived from Wikipedia, WordNet, WikiData, GeoNames, and other data sources.
- Family contains family relationships among members of a family [13].

Experimental Setup RLogic is implemented over PyTorch and trained in an end-to-end manner. Adam [14] is adopted as the optimizer. Embeddings of all predicates are uniformly initialized and no regularization is imposed on them. The detailed hyperparameter settings can be found as follows: we set embedding dimension of predicates as 1024, batch size as 512, learning rate as 0.005 and the maximum number of training epochs as 20. To fairly compare with different baseline methods, we set the parameters for all baseline methods by a grid search strategy. The best results of baseline methods are used to compared with our proposed RLogic. All the experiments are run on Tesla V100 GPUs.

B.2 Quality of Learned Rules in Terms of KG Completion Task

Experiment Setup for Transductive Link Prediction Considering that the majority of logical rule learning methods are unable to handle long rules due to the scalability issue, we thus *limit the maximum length of learned rules to be 2.* We sample 50,000, 100,000, 200,000 paths for WN18RR and FB15K237 and YAGO3-10 in order to learn rules and we select 2400 rules for each of these datasets to predict links.

Experiment Setup for Inductive Link Prediction It is important to note that comparing logical rule learning methods with KGE methods solely on transductive KG completion task is not fair. Unlike KGE methods, logical rule learning methods are not designed specifically for transductive KG completion task. The severe incompleteness of KGs usually limits the power of logical rules in transductive KG completion task. Besides, KGE methods are not capable of reasoning on unseen entities, which results in their inability in dealing with the inductive setting. To show the power of rule learning methods in inductive setting, we randomly divide the all triples in KG into training, validate and test sets with ratio of 7:1:2 *with disjoint entities.*

B.3 Quality of Learned Rules in Terms of Rule Head Prediction Task.

Dataset and Evaluation Metric. Throughout this section we use the Family dataset for demonstration purposes as it is more tangible. A set of closed paths with maximum length as 2 are sampled from Family KG to construct the training dataset. Meanwhile, 3910 rule bodies whose length ranges from 2 to 6 are taken as the test dataset. We use human annotation to annotate the rule heads for all rule bodies in test dataset for ground truth preparation. Mean Average Precision (MAP) are taken as evaluation metric as it is widely used for evaluating the ranked retrieval results.

B.4 Impact of Closed Path Sampler

To further investigate how the number of sampled closed paths affects the performance, we vary the number of sampled closed paths among $\{50, 100, 500, 1000, 5000, 10000\}$ and report performance in terms of KG completion task on Family dataset. As depicted in Figure 8, the performances increase severely when the number of sampled closed paths increases from 50 to 100. After that the

Neural-LP	RLogic
$\text{brother}(x,y) \leftarrow \text{inv\_sister}(x,y)$	$\text{brother}(x,y) \leftarrow \text{brother}(x,z) \wedge \text{sister}(z,y)$
$\text{brother}(x,y) \leftarrow \text{sister}(x,z) \wedge \text{inv\_brother}(z,y)$	$\text{brother}(x,y) \leftarrow \text{son}(x,z) \wedge \text{father}(z,y)$
$\text{brother}(x,y) \leftarrow \text{inv\_sister}(x,z) \wedge \text{inv\_sister}(z,y)$	$\text{brother}(x,y) \leftarrow \text{brother}(x,z) \wedge \text{brother}(z,y)$
$\text{wife}(x,y) \leftarrow \text{inv\_husband}(x,z) \wedge \text{inv\_husband}(z,y)$	$\text{wife}(x,y) \leftarrow \text{mother}(x,z) \wedge \text{inv\_father}(z,y)$
$\text{wife}(x,y) \leftarrow \text{inv\_husband}(x,y)$	$\text{wife}(x,y) \leftarrow \text{mother}(x,z) \wedge \text{daughter}(z,y)$
$\text{wife}(x,y) \leftarrow \text{inv\_husband}(x,z) \wedge \text{daughter}(z,y)$	$\text{wife}(x,y) \leftarrow \text{mother}(x,z) \wedge \text{son}(z,y)$
$\text{son}(x,y) \leftarrow \text{brother}(x,y) \wedge \text{inv\_mother}(z,y)$	$\text{son}(x,y) \leftarrow \text{brother}(x,z) \wedge \text{inv\_father}(z,y)$
$\text{son}(x,y) \leftarrow \text{inv\_mother}(x,z) \wedge \text{inv\_mother}(z,y)$	$\text{son}(x,y) \leftarrow \text{brother}(x,z) \wedge \text{son}(z,y)$
$\text{son}(x,y) \leftarrow \text{brother}(x,y)$	$\text{son}(x,y) \leftarrow \text{son}(x,z) \wedge \text{inv\_wife}(z,y)$

Table 8: Top 3 rules learned by Neural-LP v.s. RLogic on Family dataset.

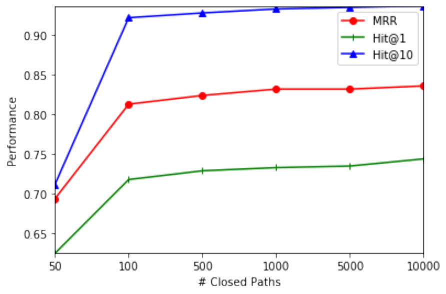


Figure 8: Performance in terms of KG completion task w.r.t. # sampled closed paths on Family dataset.

$\text{aunt}(x,y) \leftarrow \text{inv\_brother}(x,z) \wedge \text{sister}(z,y)$
$\text{mother}(x,y) \leftarrow \text{inv\_wife}(x,z) \wedge \text{inv\_daughter}(z,y)$
$\text{aunt}(x,y) \leftarrow \text{inv\_son}(x,z) \wedge \text{inv\_wife}(z,y)$
$\text{aunt}(x,y) \leftarrow \text{uncle}(x,z) \wedge \text{sister}(z,y)$
$\text{father}(x,y) \leftarrow \text{inv\_wife}(x,z) \wedge \text{niece}(z,y)$

Table 9: Low-accuracy rules learned by RLogic.

performance stay steady length. This is attractive in real-world application as a small number of sampled closed paths can already gives great performance.

B.5 Quality and Interpretability of the Rules

We also sort the rules generated by RLogic and Neural-LP on Family dataset based on their assigned confidences and show the three top rules in Table 8. Logically incorrect rules are highlighted by red while rules which do not always hold true are highlighted by blue. This experiment shows that many of the top ranked rules generated by Neural-LP are either incorrect or partially correct. The major reasoning is that Neural-LP tend to learn higher score of a meaningless rule by mistake when it shares an atom with a useful rule. Due to the introduction of deductive nature, RLogic can learn more accurate logical rules.

To better understand RLogic, we also provide rules with lowest confidences learned by RLogic in Table 9. We can observe all of these low-accuracy rules are completely logically incorrect.

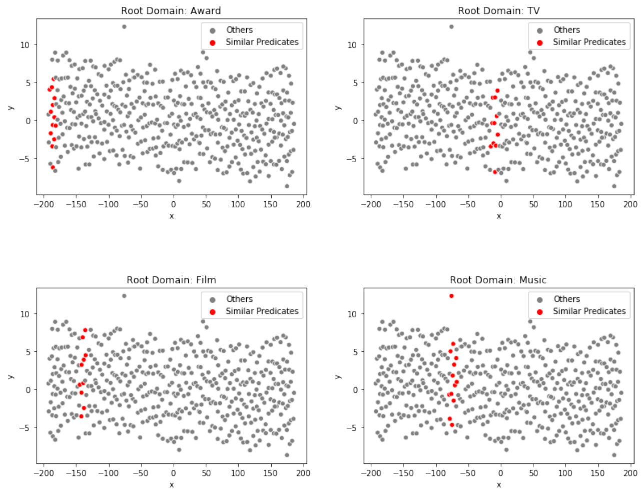


Figure 9: Visualization of Predicate Embeddings.

B.6 Visualization of Predicate Embeddings

Note that predicates in FB15K-237 dataset are grouped hierarchically into domains, we regard the predicates that share the same root domain (e.g. award, tv) as similar predicates. To show the proximity of similar predicates, we visualize the embeddings of predicates on FB15k237 dataset, which are plotted as PCA projections in Figure 9. Four different root domains, including **award**, **tv**, **film** and **music** are given. We highlight the similar predicates by red while others by grey. We can observe that the similar predicates are lined up in a vertical line, showing their proximity.