Neural-Answering Logical Queries on Knowledge Graphs

Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, Hanghang Tong
Department of Computer Science, University of Illinois at Urbana Champaign
{lihuil2,boxindu2,hengji,czhai,htong}@illinois.edu

ABSTRACT

Logical queries constitute an important subset of questions posed in knowledge graph question answering systems. Yet, effectively answering logical queries on large knowledge graphs remains a highly challenging problem. Traditional subgraph matching based methods might suffer from the noise and incompleteness of the underlying knowledge graph, often with a prolonged online response time. Recently, an alternative type of method has emerged whose key idea is to embed knowledge graph entities and the query in an embedding space so that the embedding of answer entities is close to that of the query. Compared with subgraph matching based methods, it can better handle the noisy or missing information in knowledge graph, with a faster online response. Promising as it might be, several fundamental limitations still exist, including the linear transformation assumption for modeling relations and the inability to answer complex queries with multiple variable nodes. In this paper, we propose an embedding based method (NewLook) to address these limitations. Our proposed method offers three major advantages. First (*Applicability*), it supports four types of logical operations and can answer queries with multiple variable nodes. Second (*Effectiveness*), the proposed NewLook goes beyond the linear transformation assumption, and thus consistently outperforms the existing methods. Third (*Efficiency*), compared with subgraph matching based methods, NewLook is at least 3 times faster in answering the queries; compared with the existing embedding based methods, NewLook bears a comparable or even faster online response and offline training time.

CCS CONCEPTS

• **Information systems applications** → **Data mining**; • **Artificial intelligence** → **Knowledge representation and reasoning**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Knowledge graph query; Knowledge graph embedding; Logical query embedding

ACM Reference Format:
Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, Hanghang Tong. 2021. Neural-Answering Logical Queries on Knowledge Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447548.3467375

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467375

1 INTRODUCTION

Question answering on large knowledge graphs has numerous applications. Many methods have been developed for answering different types of questions, ranging from single-hop queries [1], multi-hop queries [18], to logical queries [19]. Among them, logical queries constitutes an important subset[1] of questions posed in question answering systems. Examples of logical queries include "*Who is the spouse of Barack Obama?*" "*In 2018, who won the Turing award and where do they come from?*" Despite its key importance, it remains a difficult challenge to effectively answer logical queries on large knowledge graphs.

The traditional approaches first transform the original question into a *logical query graph* (also known as '*dependency graph*' or '*logical DAG graph*', or '*query graph*' for short) (e.g., [24, 28, 31]). For example, the aforementioned question "*Who is the spouse of Barack Obama?*" can be transformed into a logical query graph "$?\xrightarrow{isMarriedTo}$ *Barack Obama*". We can then use subgraph matching (e.g., [2, 12, 17, 23]) to find the answers from the underlying knowledge graphs. However, real-world knowledge graphs are often incomplete with noisy information [7], which makes subgraph matching based methods suffer from a number of issues, e.g., the empty-answer problem [27], the wrong answer problem [19] and so on. Besides, the intrinsic high computational complexity of subgraph matching-based methods often lead to a prolonged online response time [19].

Recently, an alternative type of method for answering logical queries has emerged (e.g., [1, 5, 7, 11, 19]). The key idea is to embed the logical query graph and the knowledge graph entities into the same low-dimensional vector space, so that the embedding of entities that answer the query graph is close to that of the query. Therefore, after the embedding is obtained, answering logical queries essentially becomes a similarity search problem in the embedding space. In this way, the reasonable answers could be found even when the knowledge graph is incomplete or noisy. What is more, compared with subgraph matching based methods, embedding based methods tend to have a faster online response.

Besides its promising results, several fundamental limitations still exist with embedding based methods. Let us elaborate using four most recent approaches as examples, including GQE [5], Query2Box [19], BetaE [20] and EmQL [21]. First, the existing methods like GQE [5] and Query2Box [19] treat the one hop transition between two nodes in the embedding space as a *linear transformation*. This makes themselves suffer from severe cascading errors [8], especially when dealing with long, complex logical queries with multiple inter-dependent paths. Second, GQE [5] and Query2Box [19] only

[1] Examples of questions which *cannot* be represented as logical queries include "*Which is the best university in computer science?*" "*How many programmers did Google hire last year?*"

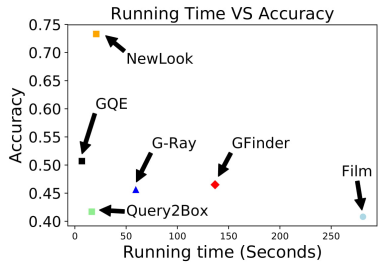


Figure 1: Accuracy and query time of different methods. Compared with the existing embedding based methods (GQE [5] and Query2box [19]), the proposed method (NewLook) is significantly better in accuracy with a comparable online query response time; compared with the subgraph matching based methods (G-Ray [23], FilM [15] and GFinder [12]), the proposed NewLook is both more accurate and faster. The results are for query '3ippu' in Figure 8 on NELL dataset. See Section 4.3 for more details.

support three logical operations, namely projection, intersection and union, and thus limit their applicability. Although BetaE [20] further supports the negation operation and EmQL [21] also supports more logical operations: relation filtering and the difference operation. Nonetheless, both methods suffer from some subtle deficiency in modeling difference operation (see Section 3.1 for more discussions).

In response, we propose an embedding based algorithm (NewLook) to answer logical queries on knowledge graphs. Compared with previous methods, our method has three distinctive advantages. First, we go beyond the linear transformation assumption and model each type of logical operation as a neural network. In this way, we can significantly mitigate the cascading errors. Second, in addition to projection, intersection and union, our method also supports a new type of logical operation, namely *difference*. Third, our method can simultaneously find the geometric embedding for multiple variable nodes in the logical query, which not only broadens the applicability of our method, but also offers a pruning method for subgraph matching based algorithms. We evaluate NewLook on three benchmark datasets and show that our method (1) outperforms both subgraph matching based methods and prior embedding based methods; and (2) is computationally efficient. See Figure 1 for a comparison.

The main contributions of the paper are

- We propose an algorithm NewLook which can not only mitigate the cascading error problem but also support more logical operations.
- we redesign the projection operation, intersection operation and loss function which greatly improve the accuracy. Moreover, NewLook can learn a box embedding for each variable node in the logical query with high accuracy.
- We perform extensive empirical evaluations to demonstrate the efficacy of our model.

2 PROBLEM DEFINITION

Table 1 gives the main notations used throughout this paper. We use upper case calligraphic font letters for graphs (e.g. $\mathcal{G}$, $\mathcal{Q}$, $\mathcal{C}$), lower case bold letters (e.g., $\mathbf{e}$, $\mathbf{b}$) for embedding vectors. A knowledge graph is denoted as $\mathcal{G} = (V, R, T)$ where $V = \{v_1, v_2, ..., v_n\}$ is the set of nodes/entities, $R = \{r_1, r_2, ..., r_m\}$ is the set of relations and T is the set of triples. Each triple in the knowledge graph can be denoted as (h, r, t) where $h \in V$ is the head (i.e., subject) of the triple, $t \in V$ is the tail (i.e., object) of the triple and $r \in R$ is the edge (i.e., relation, predicate) of the triple which connects the head h to the tail t. The knowledge graph structure information can be denoted as an $n \times n$ adjacency matrix $\mathbf{A}$ with $\mathbf{A}(i, j) = 1$ if v_i is connected with v_j, and $\mathbf{A}(i, j) = 0$ otherwise. For the computational efficiency, we randomly divide all the nodes in the knowledge graph into different groups, and use a three dimensional tensor $\mathcal{T}$ to denote the connectivity between different groups. For each relation $r_i \in R$, $\mathcal{T}(i, j, k) = 1$ if any node in group j connects with any node in group k by relation r_i. For each node v_i, we use a row one-hot vector $\mathbf{x}_i$ to denote which group it belongs to. If node v_i belongs to group j then $\mathbf{x}_i[j] = 1$, otherwise, it equals to 0. For a node set S, we can use a row multi-hot vector $\mathbf{x}_S$ to denote its group information. If any node in S belongs to group j, then $\mathbf{x}_S[j] = 1$, otherwise, it equals to 0.

Table 1: Notations and definitions

Symbols	Definition
$\mathcal{G} = (V, R, T)$	the knowledge graph
v_i	the i^{th} entity/node in knowledge graph
$\mathbf{e}_i$	the embedding of entity/node v_i
r_i	the i^{th} relation/edge in knowledge graph
$\mathcal{Q} = (U, R, L)$	a dependency graph
U	the node set of $\mathcal{Q}$
u_i	the i^{th} node in U
L	Logical operations
$\mathbf{b}_i$	the box embedding of U_i
$\mathbf{b}_i^c$	the center of $\mathbf{b}_i$
$\mathbf{b}_i^o$	the offset of $\mathbf{b}_i$
$\mathbf{p}_i$	the embedding of r_i
$\mathbf{p}_i^c$	the center embedding of $\mathbf{p}_i$
$\mathbf{p}_i^o$	the offset embedding of $\mathbf{p}_i$

Following the definition in [5], [19], [20], a first-order logic query contains a non-variable anchor entity set $\tilde{U} \subseteq V$, existentially quantified bound variables $u_1, ..., u_k$ and a single target variable $u_?$, which provides the query answer. In mathematical form, it can be defined as $q[u_?] = u_?.\exists u_1, ..., u_k : \pi_1 \vee \pi_2 \vee ... \vee \pi_n$, where $\pi_i = \varrho_{i1} \wedge \varrho_{i1} \wedge ... \wedge \varrho_{im}$ and $\varrho_{im} = \varpi(\tilde{u}_a, u_i)$ or $\varpi(u_i, u_j)$ or $\varpi(u_i, u_?)$. ϖ belongs to projection or negation [20]. For example, GQE [5] considers conjunctive logical query, Query2Box [19] considers existential positive first-order (EPFO) logical queries. In this paper, we consider logical queries which can support four kinds of logical operations: projection, intersection, union and difference. Besides this, instead of constraining a logical query only having a single target variable, we treat all non-anchor nodes as target variables.

A logical query graph (i.e., dependency graph) $\mathcal{Q} = (U, R, L)$ is the graph expression of the logical query, where the upper case letter $U = \{\tilde{U}, U_?\}$ denotes the node set, with $\tilde{U}$ as the anchor node set and $U_?$ as the variable node set (we use $\tilde{u}_j$ and u_j to index the specific node in $\tilde{U}$ and $U_?$ respectively), R denotes the relation set which is the same as the relation set of $\mathcal{G}$, and L denotes logical operations. According to the definition of first-order logic, $\mathcal{Q}$ is a Directed Acyclic Graph (DAG), with the anchor entities as the source nodes of the DAG [19]. An example of logical query graph is given in the first column of Figure 2.

Given a knowledge graph $\mathcal{G}$, a three dimensional tensor $\mathcal{T}$, the one-hot vector $\mathbf{x}_i$ for each node v_i and a logical query graph $\mathcal{Q}$, we aim to embed (1) each entity of the knowledge graph as a *point*, (2)

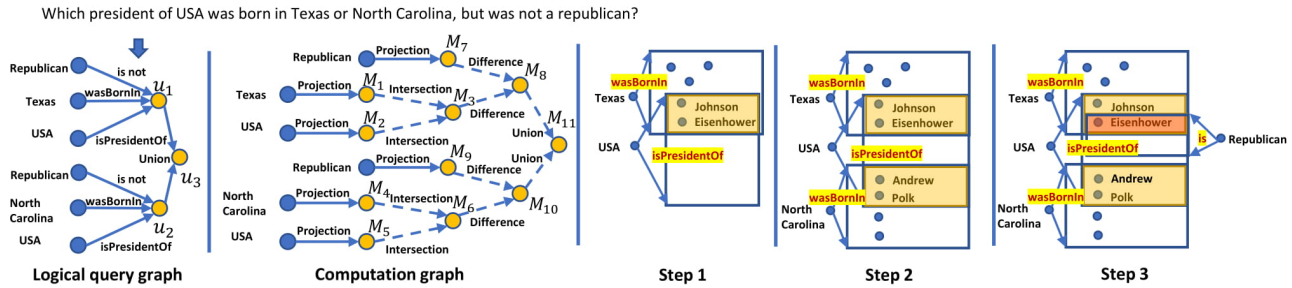


Figure 2: An illustrative example of how NewLook works.

each node of the query graph as a closed box region, namely *box embedding* [19], and (3) each relation $r \in R$ as a box embedding, in the same embedding space, so that entities that answer the query are inside or close to the boxes of the query nodes.

With box embedding, if an entity is in the answer set, it is natural to model the entity embedding to be a point inside or close to the box of the corresponding query node. We use $\mathbf{e}_i$ to denote the point embedding of $v_i \in V$ and $\mathbf{b}_j$ to denote the box embedding of node $u_j \in Q$. If node v_i is the answer of the node u_j, we hope that $\mathbf{e}_i \in \mathbf{b}_j$ or $\mathbf{e}_i$ is very close to $\mathbf{b}_j$. Formally, a box embedding is represented by its center and its offset, which is defined as follows.

$$\mathbf{b}_j = \{\mathbf{e}_k : \mathbf{b}_j^c - \mathbf{b}_j^o \leq \mathbf{e}_k \leq \mathbf{b}_j^c + \mathbf{b}_j^o\} \tag{1}$$

where $\mathbf{b}^c$ is the center of the box which represents its location, and $\mathbf{b}^o$ is the offset of the box which represents its size. An anchor node can be represented as a special box of size 0. In other words, a point embedding can be viewed as a box with its offset vector as a zero vector. Likewise, we use the lower case bold letter $\mathbf{p}$ for relation (i.e., predicate) box embedding, and its center and offset are denoted as $\mathbf{p}^c$ and $\mathbf{p}^o$ respectively. In step 1 to step 3 of Figure 2, each blue point denotes a point embedding, and each rectangle denotes a box embedding in the embedding space.

The proposed NewLook supports four logical operations in total, including projection, intersection, union and difference, which are defined as follows. Figure 3 illustrates these four logical operations.

Definition 1. Projection. Given a box embedding $\mathbf{b}_h$ and a relation r, the projection operation $\mathbb{P}$ outputs a new box embedding $\mathbf{b}_t$ where $\mathbf{b}_t$ contains the nodes connected to any node $v \in \mathbf{b}_h$ by relation r.

Definition 2. Intersection. Given a set of box embedding $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$, the intersection operation $\mathbb{I}$ outputs a new box embedding $\mathbf{b}_t$ which contains a set of nodes $\cap_{i=1}^{k} S_i$, where $S_i \subseteq \mathbf{b}_i$.

Definition 3. Difference. Given a set of box embedding $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$, the difference operation $\mathbb{D}$ outputs a new box embedding $\mathbf{b}_t$ which contains a set of nodes $S_1 - S_2 - ... - S_k$ where $S_i \subseteq \mathbf{b}_i$.

For the example in Figure 2, given the embedding of the entities `Texas`, `Republican` and relations `wasBornIn`, `is`, the difference operation generates a box embedding, which contains the embedding of all individuals who were born in `Texas` but were not a `Republican`.

Definition 4. Union. Given a set of box embedding $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$, the union operation $\mathbb{U}$ produces all the nodes that are in at least one of the k input boxes, i.e., $\cup_{i=1}^{k} S_i$ where $S_i \subseteq \mathbf{b}_i$.

Remarks. As we can see from Figure 3, the projection and intersection operations always produce a valid box. However, this is not necessarily true for either difference or union operations. In Section 3.6, we will present detailed solutions to address this issue.

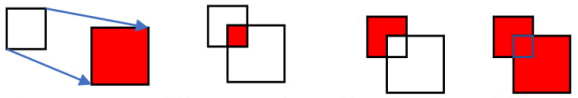


Figure 3: The illustration of four logical operations. The pink shaded area is the desirable output of the corresponding logical operation.

With these logical operations, we traverse the query graph starting from the anchor node(s) and execute the logical operations to obtain the box embedding for each variable node. A *computation graph* which can be derived from the logical query graph will guide the traverse process (e.g., to decide the orders of executing various logical operations, to manage the intermediate results). Formally a computation graph is denoted as $C = (M, L)$, where M denotes the node set and L denotes the logical operation set which is the same as the logical operation set of Q. Traversing the computation graph is equivalent to traversing the logical query graph. An example of computation graph is given in the second column of Figure 2.

With the above notations, the problem of answering logical queries on knowledge graphs boils down to how to find the point embedding of the knowledge graph entities, the box embedding of the nodes of the query graph as well as the box embedding of the relations. This can be formally defined as follows. Notice that both the three dimensional tensor $\mathcal{T}$ and the one-hot vector $\mathbf{x}_i$ for v_i can be constructed according to $\mathcal{G}$.

Problem Definition 1. *Logical query embedding.* ***Given:*** *(1) a knowledge graph $\mathcal{G} = (V, R, T)$, (2) a logical query graph Q with anchor node(s) and variable node(s);*

Output: *(1) the box embedding for each variable node in Q, (2) the point embedding for each entity $v \in \mathcal{G}$ and (3) the box embedding for each relation $r \in R$.*

3 PROPOSED MODEL

In this section, we start with the overall framework of the proposed NewLook, followed by the details of its key components.

3.1 Overall Framework

Given a logical query graph and a knowledge graph, our goal is to embed the nodes of the query graph as closed boxes and entities of the knowledge graph as points in the embedding space, so that entities that answer the query are inside or close to the boxes. To this end, we treat the query graph as a sequence of geometric logical operations, and model each logical operation as a neural network. During the training process, we learn the point embedding of the entities in knowledge graph $\mathcal{G}$ and the parameterized neural

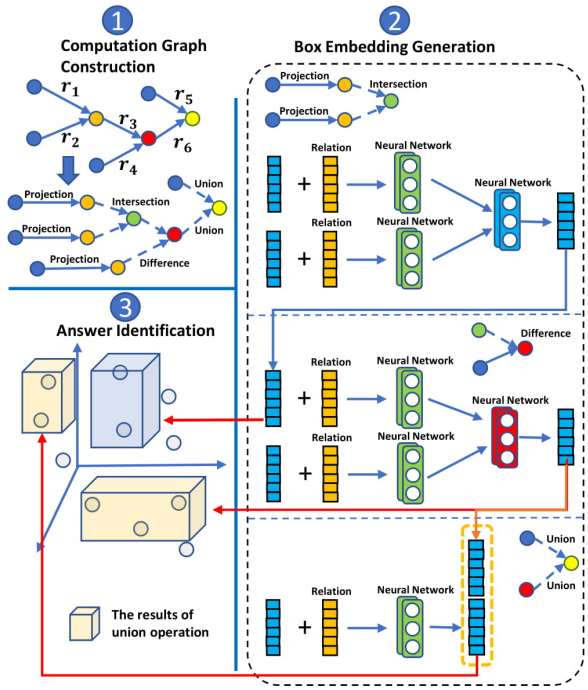


Figure 4: Framework of NewLook.

networks for logical operations at the same time. After the model is learned, given a new logical query, the box embedding of nodes of the query graph can be obtained by executing a set of logical operations. Entities that are close to the box embedding are returned as answers of the query.

A - Framework. Figure 4 presents the overall framework of the proposed NewLook. It contains three parts. In the first part (*P1. Computation Graph Construction*), given a query graph, NewLook transforms it into a *computation graph*. For each edge in the query graph, it will be substituted by a projection operation. If the in-degree of the target node is greater than one and the logical operation is intersection, all of its incoming edges will be substituted by the intersection edges. If the logical operation is difference, we do the same process but treat all incoming edges as the difference edges. If the logical operation is union, we revise the query according to Subsection 3.5. In the second part (*P2. Box Embedding Generation*), after obtaining the computation graph, NewLook traverses the query graph starting from the anchor node(s), and applies geometric logical operations one by one to generate box embedding for the variable nodes. In the third part (*P3. Answer Identification*), the generated box embedding is used to predict the likelihood that a knowledge graph entity satisfies the query, e.g., by the nearest neighbor search in the embedding space. The point embedding of the answer entities should be inside or close to their corresponding boxes, whereas the point embedding of non-answer entities should be far away from these boxes. Figure 2 gives an example to illustrate how NewLook works.

B - Relationship with the existing methods. At the high level, the three parts of NewLook are similar to the overall structure of Query2Box [19]. Indeed, some components of NewLook, including union operation and nearest neighbor search, are directly inherited from Query2Box. For the completeness of the paper, we will briefly summarize the key idea and intuitions of these components. Interested readers should refer to [19] for full details.

Compared with the existing embedding based methods (e.g., GQE [5] and Query2Box [19]), the proposed NewLook brings several important new features, including (1) modeling logical operations (e.g., projection, difference) as a nonlinear process, (2) the introduction of the difference operation, and (3) redesigning the intersection operation and loss function. As we will show in the experimental section, these new features not only bring significant performance improvement, but also broaden the applicability of the proposed method. Before diving into the details, let us highlight the advantages of these new features.

First (*Mitigating Cascading Errors*), the key of embedding a logical query into a low-dimensional space is to accurately model the impact of logical operations on the transformation of embedding of different nodes. Different logical operations (e.g. $\cap, \cup, -$) have very different properties, which in turn require different treatment. Take the projection operation as an example. It is one of the most commonly used logical operations in both knowledge graph and logical queries. Existing methods, such as TransE [1], Query2Box [19], treat projection operation as a linear transformation between two nodes in the embedding space. Despite its mathematical convenience, the real world projection operation could rarely be modeled as linear transformations, thus it would consequently make the model sub-optimal in terms of inference abilities. For example, the linear transformation could not capture some inference patterns in knowledge graph like symmetry, inversion, composition [22]. Besides this, the linear transformation assumption often leads to severe cascading errors [8], especially when dealing with long, complex queries with multiple inter-dependent paths. To mitigate these issues (see details in Subsection 3.2), in NewLook, we model the projection operation as a non-linear process by a neural network. Although EmQL [21] also has the ability to model non-linear projection operation as a top-k triple matching problem, it requires excessive off-line training time.

Second (*Supporting Difference Operation*), logical operation difference ($-$), is a commonly used logical operation in many logical queries. However, none of the existing methods, with the only possible exception of EmQL [21], supports the difference operation. Conceptually, EmQL could model the difference operation based on count-min sketch. However, it suffers when the two involved sets share some overlaps, which is quite common in logical queries. Other methods such as Gaussian embedding [6] and Beta embedding (BetaE) [20] represent a set as a distribution in the embedding space (e.g., Gaussian/Beta distribution). However, the difference between two Gaussian/Beta distributions is not Gaussian/Beta distribution anymore. Therefore, although BetaE [20] supports the negation operation[2], it does not support the difference operation. The proposed NewLook fills in this blank by modeling the difference operation as attention neural networks.

In the following subsections, we will introduce these four logical operations and how to train the entire model.

3.2 Projection Operation

Given a box embedding $\mathbf{b}_h$ of node u_h and a relation r in triple (u_h, r, u_t), the projection operation $\mathbb{P}$ outputs a new box embedding $\mathbf{b}_t$.

[2]The negation operation can be viewed as a special case of the difference operation.

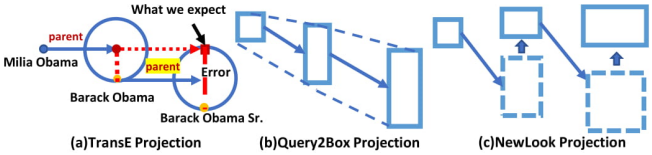


Figure 5: Comparison of different methods for modeling projection operation. Both TransE (a) and Query2Box (b) suffer from severe cascading errors. The proposed NewLook (c) models the projection as a linear transformation (the dashed boxes), followed by a nonlinear process (the solid boxes above the corresponding dashed boxes). NewLook is able to mitigate the cascading errors by adaptively adjusting the center and offset of the projection box.

A good projection operation should make the answers of u_t inside or close to the box $\mathbf{b}_t$. To be specific, suppose S_h is the node set of $\mathbf{b}_h$, the goal of the projection operation is to find another node set S_t such that $S_t = \{v_t | \exists v_h \in S_h, (v_h, r, v_t)\}$.

Many previous works have studied this problem. For example, in TransE [1], the projection operation is formulated as $\mathbf{e}_t = \mathbf{e}_h + \mathbf{p}_r^c$ where $\mathbf{e}_h$ is the embedding of the head entity and $\mathbf{e}_t$ is the embedding of the tail entity. In GQE [5], this relation is formulated as $W_r \mathbf{e}_h$ where W_r is a $d \times d$ parameter matrix for relation r. In Query2Box [19], this relation is formulated as $\mathbf{b}_t^c = \mathbf{b}_h^c + \mathbf{p}_r^c$ and $\mathbf{b}_t^o = \mathbf{b}_h^o + \mathbf{p}_r^o$. Despite the mathematical elegance of these formulations, a fundamental drawback is that they all treat the transition between the (point or box) embedding of entities as a *linear* transformation process. This makes these methods weak in terms of inference abilities. For example, the linear transformation could not capture some inference patterns in knowledge graph like symmetry, inversion, composition [22]. Moreover, the linear transformation assumption often leads to severe cascading errors [8], especially when dealing long, complex queries with multiple inter-dependent paths. For the example in Figure 5a, each point represents the embedding of an entity in the embedding space. The relation `parent` is ideally a simple horizontal translation, but each traversal introduces some noises. The red circle is where we expect `Milia`'s parent to be. The red square is where we expect `Milia`'s grandparent to be. Dotted red lines show that the error grows larger as we traverse farther away from `Milia`. Another example is shown in Figure 5b. Since Query2Box simply sums up the offset of the head entity and that of the projection operation, the offset $\mathbf{b}_t^o$ (i.e., size) of the box of the tail entity will become increasingly bigger, and thus increases the uncertainty.

In this paper, instead of modeling the projection operation as a linear transformation process, we use a non-linear neural network to learn the projection operation which can mitigate the cascading error problem. To be specific, we associate each relation $r \in R$ with a box embedding $\mathbf{p}_r = (\mathbf{p}_r^c, \mathbf{p}_r^o)$ and a group adjacency matrix $\mathcal{T}(r)$. Given a triple (u_h, r, u_t), the box embedding $\mathbf{b}_h$ and the one-hot/multi-hot vector $\mathbf{x}_{u_h}$, we first obtain an approximate box embedding $(\hat{\mathbf{b}}_t^c, \hat{\mathbf{b}}_t^o)$ of u_t by linear transformation and obtain the one-hot/multi-hot vector of u_t by $\mathbf{x}_{u_t} = \mathbb{1}[(\mathbf{x}_{u_h} \cdot \mathcal{T}(r)) > 0]$ where $\mathbb{1}[]$ is an element-wise indicator function. Then, we use a neural network to fine-tune the true box center (i.e., position) and offset (i.e., size). Our conjecture is that the true center $\mathbf{b}_t^c$ might be close to $\hat{\mathbf{b}}_t^c$ and the true offset $\mathbf{b}_t^o$ should be dependent on $\hat{\mathbf{b}}_t^c$, $\hat{\mathbf{b}}_t^o$ and $\mathbf{x}_{u_t}$. Therefore, we treat $\hat{\mathbf{b}}_t^c$, $\hat{\mathbf{b}}_t^o$ and $\mathbf{x}_{u_t}$ as the input of a neural network, to obtain the true center $\mathbf{b}_t^c$ and the true offset $\mathbf{b}_t^o$ for node u_t. The neural network is defined as follows.[3]

$$\begin{aligned} \hat{\mathbf{b}}_t^c &= \mathbf{b}_h^c + \mathbf{p}_r^c & \hat{\mathbf{b}}_t^o &= \mathbf{p}_r^o \\ \mathbf{z}_1 &= \text{MLP}(\hat{\mathbf{b}}_t^c) & \mathbf{z}_2 &= \text{MLP}(\hat{\mathbf{b}}_t^o) \\ \mathbf{b}_t^c &= \text{MLP}(\mathbf{z}_1 || \mathbf{z}_2 || \mathbf{x}_{u_t}) & \mathbf{b}_t^o &= \text{MLP}(\mathbf{z}_1 || \mathbf{z}_2 || \mathbf{x}_{u_t}) \end{aligned} \tag{2}$$

where MLP() is a multi-layer perceptron, and || represents concatenation. By modeling the projection operation as a nonlinear process, we can adaptively adjust the center and offset of the new box, which helps mitigate the cascading error problem.

3.3 Intersection Operation

Figure 6: An illustration of intersection operation. Given three boxes $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ on the left, whose intersection is empty in the training set but not in the ground truth. A neural network approach based on attention mechanism and deepset can result in a reasonable box $\mathbf{b}_4$ on the right.

Intersection is a widely used logic operation. Given a set of box embedding $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$ and a set of one-hot/multi-hot vectors $\{\mathbf{x}_{u_1}, \mathbf{x}_{u_2}, ..., \mathbf{x}_{u_k}\}$ for $\{u_1, u_2, ..., u_k\}$, the intersection operation aims to output a new box embedding $\mathbf{b}_t$ which contains the intersection of node sets $\cap_{i=1}^k S_i$, where $S_i \subseteq \mathbf{b}_i$.

If we directly calculate the overlap among all input boxes, it might result in an empty box when the input boxes do not share a common area in the training set, even if such an overlapped area might exist in the ground truth. To address this issue, Query2Box [19] designed an ingenious solution by using an attention neural network to learn the box center of the intersection operation, together with a deepset [29] neural network to learn its offset. Figure 6 presents an illustrative example. Like Query2Box [19], we model the intersection as an attention neural network. However, instead of simply using the center and offset information of each box, we further leverage the one-hot/multi-hot vector information of each box. The one-hot/multi-hot vector for u_t can be calculated as $\mathbf{x}_{u_t} = \mathbf{x}_{u_1} \odot \mathbf{x}_{u_2} \odot ... \odot \mathbf{x}_{u_k}$ where $\odot$ is the element-wise product. If $\mathbf{x}_{u_i}$ is similar with $\mathbf{x}_{u_t}$, box b_i should have a high attention score. More specifically, the mathematical formulas of intersection operation are defined as follows.

$$\begin{aligned} \mathbf{z}_i &= \frac{1}{\left\| \text{Relu}(\mathbf{x}_{u_i} - \mathbf{x}_{u_t}) \right\|_1} & \hat{\mathbf{b}}^o &= \text{Min}(\{\mathbf{b}_1^o, ..., \mathbf{b}_k^o\}) \\ \mathbf{a}_i &= \frac{\exp(\mathbf{z}_i \text{MLP}(\mathbf{b}_i^c || \mathbf{b}_i^o))}{\sum_{j=1}^k \exp(\mathbf{z}_k \text{MLP}(\mathbf{b}_k^c || \mathbf{b}_k^o))} & \mathbf{b}_t^c &= \sum_{i=1}^k \mathbf{a}_i \odot \mathbf{b}_i^c \\ \mathbf{w} &= \text{MLP}(\text{Mean}(\sum_{i=1}^k \text{MLP}(\mathbf{b}_i^c || \mathbf{b}_i^o))) & \mathbf{b}_t^o &= \hat{\mathbf{b}}^o \odot \sigma(\mathbf{w}) \end{aligned} \tag{3}$$

[3] An alternative choice is to set the approximate offset $\hat{\mathbf{b}}_t^o = \mathbf{b}_h^o + \mathbf{p}_r^o$. However, we found in the experiments that it does not perform as good as setting $\hat{\mathbf{b}}_t^o = \mathbf{p}_r^o$.

where $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$ are the input boxes, $\mathbf{b}_t$ is the output box, $\odot$ is the dimension-wise product, MLP() is a multi-layer perceptron, Mean() is the dimension-wise mean, Min() is the dimension-wise min and $\sigma()$ is the sigmoid function.

3.4 Difference Operation

Logical difference operation is another very useful logic operation. It aims to answer the question like "*Who won the Turing award in 2018 but was not born in France?*" In this paper, we model the difference operation as an attention neural network on box embedding. Given a set of box embedding $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k$, the difference operation will output a new box embedding $\mathbf{b}_t$ which contains all the entities belonging to $\mathbf{b}_1$ but not belonging to $\{\mathbf{b}_2, \mathbf{b}_3, ..., \mathbf{b}_k\}$.

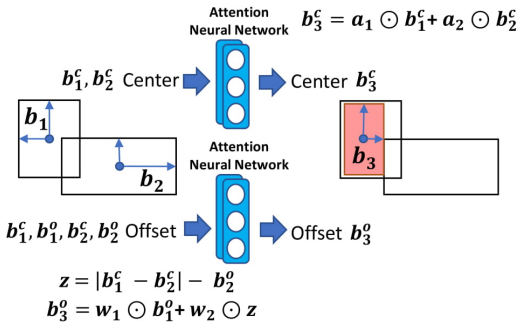


Figure 7: An illustration of difference operation. The proposed NewLook uses two attention networks to learn the center and offset of the box embedding $\mathbf{b}_3$, with the help of $\mathbf{z}$ to indicate if and how the two input boxes are overlapped with each other.

When modeling the difference of $\mathbf{b}_1$ and $\{\mathbf{b}_2, \mathbf{b}_3, ..., \mathbf{b}_k\}$, one naive way is to subtract the overlapping area from $\mathbf{b}_1$. However, the result of the difference, in general, is not a box (See Figure 3 for an example). Moreover, if $\mathbf{b}_1$ is entirely inside the overlapping area during the training process, it will result in an empty box. This will in turn prevent the gradient from flowing to the model [10], even if $\mathbf{b}_1$ is not entirely inside the overlapping area in the ground truth.

Another possible way to model the difference of $\mathbf{b}_1$ and $\{\mathbf{b}_2, \mathbf{b}_3, ..., \mathbf{b}_k\}$ is to follow the similar idea as intersection operation. That is, we could use an attention neural network to learn the center of $\mathbf{b}_t$, and use a deepset model to learn the offset of $\mathbf{b}_t$. However, the difference operation bears an important subtlety from the intersection operation, in that the difference operation is *asymmetric* (i.e., $\mathbf{b}_1 - \mathbf{b}_2 \neq \mathbf{b}_2 - \mathbf{b}_1$). Since deepset treats all inputs in a symmetric way, which means that $deepset(\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k) = deepset(\mathbf{b}_2, \mathbf{b}_1, ..., \mathbf{b}_k)$. This would make the offset of $(\mathbf{b}_1 - \mathbf{b}_2 - ... - \mathbf{b}_k)$ be equal to the offset of $(\mathbf{b}_2 - \mathbf{b}_1 - ... - \mathbf{b}_k)$, which is not necessarily true. For the example in Figure 7, the offset of $(\mathbf{b}_1 - \mathbf{b}_2)$ is much smaller than the offset of $(\mathbf{b}_2 - \mathbf{b}_1)$.

To address this issue, we consider the following two situations when calculating the offset of the resulting box $\mathbf{b}_t$. First, if there is no overlap between $\mathbf{b}_1$ and $\mathbf{b}_i$, for any $2 \leq i \leq k$, or $\mathbf{b}_1$ is partially overlapped with $\mathbf{b}_i$, the offset of $\mathbf{b}_t$ should be non-negative. Second, if $\mathbf{b}_1$ is entirely inside $\mathbf{b}_i$, the offset of $\mathbf{b}_t$ should be zero (i.e., the difference operation will result in an empty box). In order to encode this kind of information in neural network, we use $\mathbf{z}_i = |\mathbf{b}_1^c - \mathbf{b}_i^c| + \mathbf{b}_1^o - \mathbf{b}_i^o$ to measure whether $\mathbf{b}_1$ is entirely inside $\mathbf{b}_i$ or not. If $\mathbf{z}_i$ is (dimension-wisely) less than 0, it means $\mathbf{b}_1$ is entirely inside $\mathbf{b}_i$. Otherwise, $\mathbf{b}_1$ is either partially overlapped with $\mathbf{b}_i$, or there is no overlap between the two. More specifically, NewLook models the difference operation of a set of boxes via two attention neural networks which are defined as follows. See Figure 7 for an illustration.

$$
\begin{aligned}
\mathbf{t} &= \exp(\text{MLP}(\mathbf{b}_1^o)) & \mathbf{a}_i &= \frac{\exp(\text{MLP}(\mathbf{b}_i^c)))}{\sum_{i=1}^k \exp(\text{MLP}(\mathbf{b}_i^c))} \\
\mathbf{z}_i &= |\mathbf{b}_1^c - \mathbf{b}_i^c| + \mathbf{b}_1^o - \mathbf{b}_i^o \ (i = 2, .., k) & & \\
\mathbf{w}_i &= \frac{\exp(\text{MLP}(\mathbf{z}_i))}{\mathbf{t} + \sum_{i=2}^k \exp(\text{MLP}(\mathbf{z}_i))} & \mathbf{b}_t^c &= \sum_{i=1}^k \mathbf{a}_k \odot \mathbf{b}_k^c \\
\mathbf{w}_1 &= \frac{\mathbf{t}}{\mathbf{t} + \sum_{i=2}^k \exp(\text{MLP}(\mathbf{z}_i))} & \mathbf{b}_t^o &= \sum_{i=1}^k \mathbf{w}_i \odot \mathbf{b}_i^o
\end{aligned}
\tag{4}
$$

where MLP() is a multi-layer perceptron, and $\odot$ is the dimension-wise product. In BetaE [20], they proposed a new logical operation: negation. The negation of a set usually contains a huge mount of entities. As a by-product, we can directly reuse the difference operation to model the negation operation, i.e., $\neg A = (I - A)$ where I is the whole set.

3.5 Union Operation

Different from the above three logical operations, the union of multiple input boxes might not be a single box. For example, in step 3 of Figure 2, the two yellow boxes are located in distant places in the embedding space and do no overlap with each other. If we model their union as a single box, it will contain a large false positive area. Thanks to [19], *any* union operation in the query graph can be transformed to the last step. This means that if the union operation occurs somewhere in the computation graph, we can revise its logic and move the union operation to the last step. For the example in Figure 2, its original query is ((u$_?$, `isPresidentOf`, `USA`) - (u$_?$, `is`, `Republican`)) ∩ ((u$_?$, `wasBornIn`, `Texas`) ∪ (u$_?$, `wasBornIn`, `North Carolina`)). We can transform it into an equivalent query as ((u$_?$, `wasBornIn`, `Texas`) ∩ (u$_?$, `isPresidentOf`, `USA`) - (u$_?$, `is`, `Republican`)) ∪ ((u$_?$, `wasBornIn`, `North Carolina`) ∩ (u$_?$, `isPresidentOf`, `USA`) - (u$_?$, `is`, `Republican`)). Therefore, if we want to find the answering nodes for the union of multiple boxes, we only need to find the answers for each of these boxes and take the union of the answers.

3.6 Training

During the training process, we generate a set of queries together with their answers, and then learn entity embedding, relation embedding and geometric logical operations at the same time. We use the negative sampling [14] to optimize the model. The goal is to minimize the loss function which is defined as follows.

$$
\begin{aligned}
L = \sum_{u_j \in U_?} [&- \log \sigma(\gamma - d(v, \mathbf{b}_j) - \lambda * ||\mathbf{Relu}(\mathbf{x}_v - \mathbf{x}_{u_j})||_1) \\
&- \frac{1}{n} \sum_{m=1}^{n} \log \sigma(\lambda * ||\mathbf{Relu}(\mathbf{x}_{m'} - \mathbf{x}_{u_j})||_1 + d(v_{m'}, \mathbf{b}_j) - \gamma)]
\end{aligned}
\tag{5}
$$

where $d(v, \mathbf{b}_j)$ is the distance between an entity embedding vector $\mathbf{e}_v$ and a box embedding $\mathbf{b}_j$ of a variable node u_j, γ is a fixed scalar margin, v is a positive entity (i.e., the answer to the node u_j), $v_{m'}$ is the m^{th} negative node (m' is its index), and n is the negative sample size. In order to distinguish the nodes inside the box from the nodes outside the box, the distance function $d(v, \mathbf{b}_j)$ splits the distance into two parts, including the within box distance and outside box distance, and it down-weights the within-box distance by a weighting parameter $0 < \alpha < 1$. In this way, as long as the

nodes are inside the box, they will be regarded as "close enough" to the box center. The term $\lambda * ||\mathbf{Relu}(\mathbf{x}_v - \mathbf{x}_{u_j})||_1$ is used to measure the distance between the one-hot vector $\mathbf{x}_v$ and one-hot/multi-hot vector $\mathbf{x}_{u_j}$. The full details of the distance function are in [19], which can also be found in Appendix.

4 EXPERIMENTS

In this section, we conduct empirical studies to evaluate the performance of the proposed NewLook method.

4.1 Datasets and Baseline Methods

We run the experiments on three commonly used knowledge graphs, including FB15k, FB15k-237 and NELL. FB15k is a widely used knowledge graph which is a subset of FreeBase [1]. FB15k-237 is a subset of FB15k by removing the duplicated relations. NELL is a knowledge graph used by Query2Box [19] and GQE [5]. Table 4 summarizes the statistics of these three datasets. We compare the proposed NewLook algorithm against the state-of-the-art embedding base methods including GQE [5], Query2Box [19], BetaE [20] and EmQL [21] [4] , and subgraph matching based algorithms including FilM [15], GFinder [12] and G-Ray [23]. More details about datasets and baseline algorithms can be found in Appendix.

We use 18 different query structures in the experiments in total, which can be divided into two groups. For each query structure, we generate multiple query graphs. The first group (Figure 8a) contains 12 query structures, which are used to evaluate queries with a single target variable node. This is the same setting as the existing embedding based methods for answering logical queries. In detail, the query structures 1p, 2p, 3p, 2i, 3i, 2d and 3d are used for training, validation and testing the model, while query structures ip, pi, 2u, up, and dp are only used in the validation and test phrases in order to evaluate the inductive ability of the model. For a fair comparison, we follow the exact same experimental setting as Query2Box and GQE which only predict the answers for a single target node (the red node in Figure 8a). The second group of query structures are for queries with multi-variable nodes, which include six larger query structures (Figure 8b). Our goal is to find the candidate answers for each of the variable nodes in the query structure. We compare NewLook with subgraph matching methods to show the power of embedding methods. The details on how to generate the training data, testing data and validation data are given in Appendix.

In the experiments, we set embedding dimensionality $d = 400$, $\gamma = 24$ (Eq. (5)), and $\alpha = 0.2$ (Eq. (7)). We train all types of training structures jointly. In each iteration, we sample a minibatch of 512 queries, and the negative sampling number is set to 128. We optimize the loss in Eq. (5) using Adam Optimizer [9] with a learning rate of 0.0001. We train all models for 50,000 iterations.

4.2 Queries with A Single Target Variable Node

Here, we compare NewLook with Query2Box and GQE to test their performance in answering queries with a single target variable node. Following the same setting as Query2Box, We use 1p, 2p, 3p, 2i, 3i, 2d and 3d for training, validation and testing, and use ip, pi, 2u, up, and dp to test the generalization ability of the model to unseen queries. Since the original Query2Box and GQE do not have the ability to handle difference operation, we implement the difference operation in both Query2Box and GQE.

4.2.1 Evaluation Protocol. Given a query q, we use A_q to denote the answer set of q. The rank of v is denoted as $\text{Rank}(v)$. We use Hits at K (H@K) as the evaluation metric for answering query q:

$$\text{H@K}(q) = \frac{1}{|A_q|} \sum_{v \in A_q} \mathbb{1}(\text{Rank}(v) \leq K) \tag{6}$$

where $\mathbb{1}[x \leq K]$ is an indicator function and it equals to 1 if $x \leq K$, and 0 otherwise.

The overall accuracy for each query structure is calculated by averaging Eq. 6 over all queries with the same structure. The same evaluation metric is used for training, validation and testing.

4.2.2 Main Results. As shown in Table 2, NewLook significantly and consistently outperforms both Query2Box and GQE across all the query structures, including those not seen during the training stage as well as those with union or difference operations. On average, we obtain 25% higher H@1 and 17% higher H@3 than the best competitor on FB15k dataset, and 30% higher H@1, 27% higher H@3 and 20% higher H@10 than the best competitor on NELL dataset. For some query structures (e.g. 1p, 2u), we obtain 60% higher H@1 than the best competitor on NELL dataset. For query graph 3p, we obtain more than 15% higher H@1, H@3 and H@10 on average for all three datasets. This is consistent with our analysis in Section 3.2, that is, the linear transformation behind Query2Box and GQE leads to cascading errors. What is more, NewLook performs well on both queries with the same structure as the training queries and new query structures unseen during training, which demonstrates that NewLook generalizes well within and beyond query structures.

4.2.3 BetaE vs. EmQL vs. NewLook. Here, we compare NewLook with BetaE and EmQL [5]. Following the same setting as BetaE and EmQL, we use 1p, 2p, 3p, 2i, and 3i for training, validation and testing, and use ip, pi, 2u, and up to test the generalization ability of the model to unseen queries. Since Mean Reciprocal Rank (MRR) is the only common metric used by both BetaE and EmQL in their papers, for a fair comparison, we use this metric here. MRR is defined as $\frac{1}{|A_q|} \sum_{v \in A_q} \frac{1}{\text{Rank}(v)}$ (the higher the better). Figure 9 shows the Average MRR of different methods on different datasets w.r.t. the offline training time. As we can see, NewLook has the highest average MRR and the shortest offline training time.

4.3 Queries with Multi-variable Nodes

For a query with multi-variable nodes, we aim to find the answers for all variable nodes in the query graph, which is more challenging compared with queries with a single target node. Since the original GQE and Query2Box can only find the answers for a single target node, for a fair comparison, we have extended GQE and Query2Box to support multi-variable nodes.

We use six query structures in the experiments which are shown in Figure 8b. For each query structure, we generate 100 query graphs. When instantiating queries, given a query structure, we use subgraph matching algorithm to find multiple subgraphs in the data graph, and merge all the subgraphs if they share the same anchor nodes. We then start from these anchor nodes to execute the subgraph matching process and find all the answers for each variable

[4] In theory, EmQL can model the difference operation. However, we could not find this function in its current official code. Therefore, the comparison with EmQL in the context of the difference operation is omitted.

[5] We skip the comparison with EmQL-Entailment[21], since it requires the entire knowledge graph for training and does not work in the inductive setting.

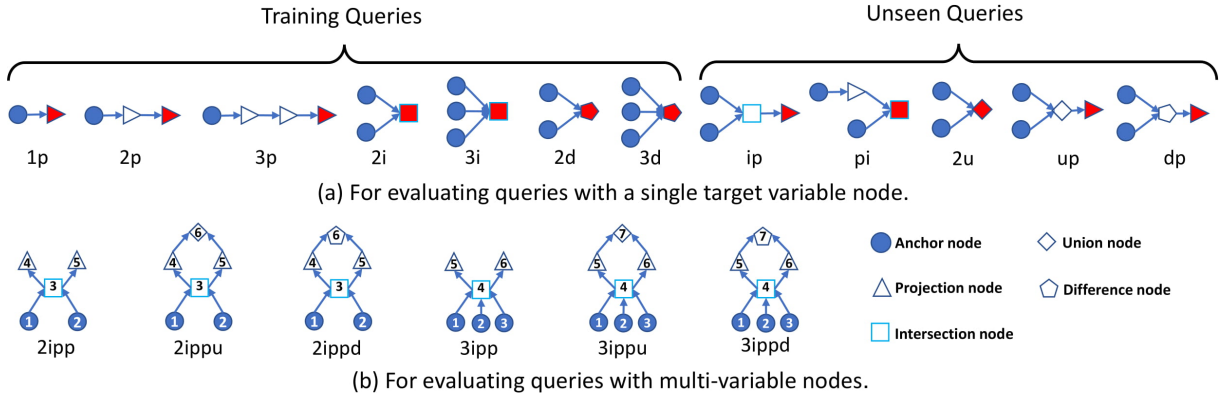


Figure 8: Query structures used in the experiments, where 'p', 'i', 'd' and 'u' stand for 'projection', 'intersection', 'difference' and 'union', respectively. Numbers before 'i', 'd' and 'u' denote their input size (i.e., the number of anchor nodes). Number before 'p' denotes the length of the path (i.e., the number of projection operations).

Table 2: Answering queries with a single target variable node. NLK refers to NewLook, Q2B refers to Query2Box.

Query	1p			2p			3p			2i			3i			ip			pi			2u			up			2d			3d			dp			Average		
Method	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk	GQE	Q2B	NLk
FB15k																																							
Hits@1	0.31	0.48	**0.81**	0.15	0.23	**0.51**	0.11	0.15	**0.39**	0.20	0.32	**0.51**	0.27	0.41	**0.69**	0.07	0.10	**0.21**	0.12	0.19	**0.48**	0.14	0.25	**0.81**	0.11	0.17	**0.25**	0.31	0.49	**0.88**	0.12	0.15	**0.38**	0.17	0.23	**0.34**	0.17	0.27	**0.52**
Hits@3	0.69	0.78	**0.88**	0.32	0.38	**0.64**	0.22	0.26	**0.51**	0.44	0.58	**0.72**	0.55	0.69	**0.78**	0.13	0.18	**0.31**	0.27	0.37	**0.61**	0.43	0.59	**0.94**	0.24	0.29	**0.37**	0.66	0.77	**0.95**	0.36	0.32	**0.54**	0.36	0.33	**0.46**	0.39	0.47	**0.64**
Hits@10	0.85	0.90	**0.93**	0.48	0.54	**0.75**	0.35	0.40	**0.64**	0.63	0.75	**0.80**	0.74	0.84	**0.89**	0.24	0.30	**0.43**	0.44	0.54	**0.74**	0.68	0.81	**0.98**	0.40	0.46	**0.51**	0.83	0.90	**0.97**	0.44	0.46	**0.69**	0.52	0.46	**0.59**	0.56	0.62	**0.74**
FB15k-237																																							
Hits@1	0.20	0.27	**0.68**	0.10	0.13	**0.30**	0.07	0.09	**0.19**	0.10	0.14	**0.47**	0.16	0.22	**0.57**	0.04	0.05	**0.08**	0.06	0.09	**0.28**	0.05	0.07	**0.49**	0.06	0.09	**0.15**	0.29	0.40	**0.76**	0.16	0.26	**0.29**	0.15	0.16	**0.27**	0.12	0.16	**0.37**
Hits@3	0.39	0.45	**0.85**	0.19	0.22	**0.43**	0.13	0.16	**0.31**	0.25	0.31	**0.71**	0.36	0.43	**0.71**	0.07	0.10	**0.16**	0.15	0.19	**0.41**	0.15	0.22	**0.70**	0.14	0.17	**0.24**	0.54	0.64	**0.86**	0.38	0.44	**0.46**	0.28	0.26	**0.39**	0.25	0.29	**0.52**
Hits@10	0.57	0.63	**0.94**	0.32	0.36	**0.59**	0.24	0.28	**0.45**	0.43	0.50	**0.81**	0.54	0.61	**0.81**	0.15	0.19	**0.25**	0.27	0.32	**0.54**	0.33	0.43	**0.86**	0.27	0.31	**0.37**	0.72	0.79	**0.94**	0.57	0.62	**0.64**	0.43	0.39	**0.53**	0.39	0.45	**0.64**
NELL																																							
Hits@1	0.13	0.20	**0.81**	0.08	0.11	**0.45**	0.08	0.10	**0.33**	0.09	0.14	**0.60**	0.14	0.24	**0.66**	0.04	0.05	**0.13**	0.08	0.10	**0.33**	0.03	0.07	**0.68**	0.04	0.06	**0.23**	0.20	0.29	**0.88**	0.16	0.28	**0.36**	0.19	0.19	**0.46**	0.11	0.14	**0.49**
Hits@3	0.44	0.53	**0.92**	0.20	0.23	**0.34**	0.19	0.21	**0.48**	0.27	0.33	**0.74**	0.36	0.45	**0.80**	0.08	0.11	**0.21**	0.17	0.19	**0.46**	0.22	0.33	**0.85**	0.12	0.13	**0.35**	0.55	0.69	**0.95**	0.41	0.43	**0.54**	0.38	0.33	**0.60**	0.28	0.33	**0.60**
Hits@10	0.62	0.70	**0.96**	0.35	0.39	**0.48**	0.30	0.34	**0.63**	0.47	0.55	**0.84**	0.58	0.66	**0.89**	0.17	0.20	**0.32**	0.28	0.32	**0.59**	0.44	0.56	**0.93**	0.27	0.29	**0.47**	0.72	0.82	**0.98**	0.63	0.64	**0.71**	0.54	0.49	**0.71**	0.45	0.49	**0.71**

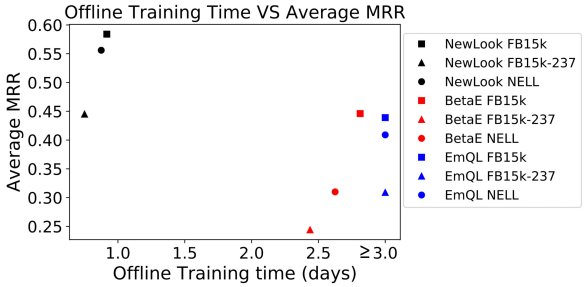


Figure 9: Average MRR results on the Query2Box datasets.

Table 3: Answering queries with multi-variable nodes.

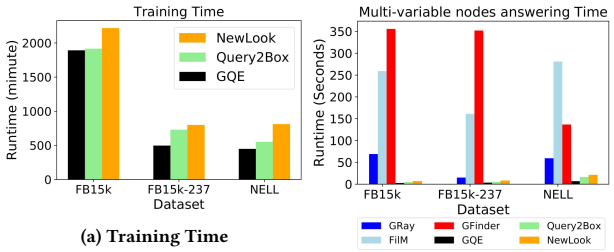
Method	GQE	Q2B	GRay	FilM	GFinder	NewLook
2ipp	0.550	0.481	0.554	0.566	0.638	**0.720**
2ippu	0.592	0.426	0.505	0.583	0.631	**0.761**
2ippd	0.462	0.435	0.452	0.637	**0.652**	0.641
3ipp	0.447	0.442	0.513	0.394	0.437	**0.688**
3ippu	0.507	0.417	0.456	0.408	0.465	**0.733**
3ippd	0.447	0.395	0.421	0.423	0.482	**0.634**
Average	0.500	0.432	0.483	0.501	0.550	**0.696**

node in the query. In order to test the generalization ability of different models (e.g. GQE, Query2Box and NewLook), we train them on the following query structures, including 1p, 2p, 3p, 2i, 3i, 2d, and id.

4.3.1 Evaluation Protocol. We use the following metric to measure the accuracy $Accuracy = \frac{F}{\min(N,10)}$, where F is the number of qualifying nodes in the top 10 candidate nodes, and N is the number of nodes in the ground truth. A qualifying node is the one which can form a connected subgraph with other qualifying nodes of other variable nodes. We normalize F by $\min(N, 10)$ because the total number of answers of a variable node may be less than 10 in the ground truth.

For GQE, Query2Box and NewLook, given a variable node u_i in the query graph, for each node $v \in V$, we use $d(v, \mathbf{b}_i)$ to rank v. We then select the top-10 entities which have the smallest distance as the candidate answers. For G-Ray, we treat the square node (i.e., the intersection node) as the seed node, and set the size of the seed list as 10. For each seed node, we use G-Ray to find the subgraph, and merge all the results together to calculate the accuracy. For FilM and GFinder, we randomly select 10 nodes from their candidate lists, and calculate the corresponding accuracy.

Table 3 shows the average accuracy of different methods on NELL dataset. For each query structure, we calculate the accuracy for each variable node, then we take the average of all variable nodes of a query structure, and treat it as the overall accuracy of multi-variable nodes. As we can see, NewLook has the highest accuracy in all but one query structures. On average, NewLook



(a) Training Time

(b) Testing Time

Figure 10: Running time of different methods.

obtains more than 10% higher accuracy than the best competitor. For query 3ipp, 3ippu, and 3ippd, NewLook obtains more than 15% higher accuracy than the best competitor.

Figure 10 shows the offline training time and online query time of different methods. For the query time, we run each structure separately, and then take the average running time of these six query structures. For each query structure, there are 100 queries and we count the total running time. Since GFinder, FilM and GRay do not require offline training, we omit them in Figure 10a. As we can see, on average, GQE has the shortest online query time (about 5 seconds), whereas GFinder has the longest online query time (about 281 seconds).

We found that most of the query time of G-Ray is spent on calculating the node proximity matrix; most of the query time of GFinder is spent on parsing data graph and building indexing for the data graph. For FilM, its running time scales near linearly w.r.t the number of query graphs. For GQE, Query2Box and NewLook, most of their query time is spent on answer identification which needs to sort all the nodes in the data graph according to their distance to the point or box embedding of the query node. This suggests that there might be room to further speedup the online query response of NewLook, say based on efficient indexing techniques (e.g., k-d-tree) or locality sensitive hashing.

Figure 1 shows the trade-off between the accuracy and online query time of different algorithms on a single query structure (i.e., 3ippu on NELL dataset). As we can see, (1) embedding based methods (e.g., NewLook, GQE and Query2Box) are much faster than subgraph matching based methods (e.g., G-Ray, Film and GFinder), and (2) the proposed NewLook achieves the highest accuracy with a comparable online response time with GQE and Query2Box. More experimental results and ablation study can be found in Appendix.

5 RELATED WORK

Subgraph matching based knowledge graph question answering has been studied for a long time. Some methods use semantic parsing [28] to transform the natural language query into a query graph or executable queries such as SPARQL [30]. Other methods transform the input question into a structured query by employing templates [24, 31]. After the query graph is obtained, many subgraph matching methods can be used to find the results (e.g. [2, 12, 17, 23]).

Knowledge graph embedding aims to embed the components of a knowledge graph (e.g. entities and relations) into a low-dimensional space. Most relevant work includes single-hop reasoning [1, 11, 22], multi-hop reasoning [8, 18] and geometric embedding, such as regions (e.g., box, sphere) [3, 25], probability (Gaussian distribution) [6, 26], and so on.

Graph embedding aims to embed the nodes in the graph in low dimension embedding space so that similar nodes will be close to each other in the embedding space. Some works focus on learning the embedding for static graphs [4, 16], while other focus on dynamic graphs [13].

6 CONCLUSION

In this paper, we present an embedding based algorithm NewLook to answer complex logical queries on knowledge graphs. NewLook supports 4 types of logical operations and can answer queries with multiple variable nodes. Experimental results show that NewLook outperforms both subgraph matching based methods and embedding based methods, and it is computationally efficient. Further directions include integrating subgraph matching based and embedding based methods for answering logical queries.

7 ACKNOWLEDGEMENT

This work is supported by National Science Foundation under grant No. 1947135, by the United States Air Force and DARPA under contract number FA8750-17-C-0153 [6], and IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. The content of the information in this document does not necessarily reflect the position or the policy of the Government or Amazon, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] N Usunier A Bordes. [n.d.]. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26.*
[2] N Cao B Du, S Zhang and H Tong. 2017. FIRST: Fast Interactive Attributed Subgraph Matching. In *Proceedings of the 23rd ACM SIGKDD.*
[3] Katrin Erk. 2009. Representing Words as Regions in Vector Space. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning.*
[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. arXiv:1607.00653 [cs.SI]
[5] W Hamilton, P Bajaj, M Zitnik, D Jurafsky, and J Leskovec. 2018. Embedding Logical Queries on Knowledge Graphs *(NIPS'18).* 2030–2041.
[6] S He, K Liu, G Ji, and J Zhao. 2015. Learning to Represent Knowledge Graphs with Gaussian Embedding *(CIKM '15).*
[7] Sanghyun Hong, Noseong Park, Tanmoy Chakraborty, Hyunjoong Kang, and Soonhyun Kwon. 2018. PAGE: Answering Graph Pattern Queries via Knowledge Graph Embedding. In *Big Data – BigData 2018.*
[8] P Liang K Guu, J Miller. [n.d.]. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.*
[9] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]
[10] X Li, L Vilnis, D Zhang, M Boratko, and A McCallum. [n.d.]. Smoothing the Geometry of Probabilistic Box Embeddings. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.*
[11] Y Lin, Z Liu, M Sun, Y Liu, and X Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence.*
[12] L. Liu, B. Du, J. xu, and H. Tong. 2019. G-Finder: Approximate Attributed Subgraph Matching. In *2019 IEEE International Conference on Big Data (Big Data).* 513–522.
[13] Zhining Liu, Dawei Zhou, Yada Zhu, Jinjie Gu, and Jingrui He. 2020. Towards Fine-Grained Temporal Network Representation via Time-Reinforced Random Walk. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (Apr. 2020), 4973–4980. https://doi.org/10.1609/aaai.v34i04.5936
[14] T Mikolov, I Sutskever, K Chen, G Corrado, and J Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems.*

[6] Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

[15] J. D. Moorman, Q. Chen, T. K. Tu, Z. M. Boyd, and A. L. Bertozzi. 2018. Filtering Methods for Subgraph Matching on Multiplex Networks. In *2018 IEEE International Conference on Big Data (Big Data)*. 3980–3985.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (Aug 2014). https://doi.org/10.1145/2623330.2623732

[17] Robert Pienta, Acar Tamersoy, Hanghang Tong, and Duen Horng Chau. 2014. MAGE: Matching approximate patterns in richly-attributed graphs. *2014 IEEE International Conference on Big Data (Big Data)* (2014), 585–590.

[18] A Neelakantan R Das and A McCallum. 2017. Chains of Reasoning over Entities, Relations, and Text using Recurrent Neural Networks. In *The 15th Conference of the European Chapter of the Association for Computational Linguistics*.

[19] Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings. In *International Conference on Learning Representations*.

[20] Hongyu Ren and Jure Leskovec. 2020. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. arXiv:2010.11465 [cs.AI]

[21] Haitian Sun, Andrew O. Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W. Cohen. 2021. Faithful Embeddings for Knowledge Base Queries. arXiv:2004.03658 [cs.LG]

[22] Z Sun, Z Deng, J Nie, and J Tang. [n.d.]. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *CoRR* ([n. d.]).

[23] H Tong, C Faloutsos, B Gallagher, and T Eliassi-Rad. 2007. Fast Best-Effort Pattern Matching in Large Attributed Graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*.

[24] C Unger, L Bühmann, J Lehmann, A Ngonga N, D Gerber, and P Cimiano. [n.d.]. Template-Based Question Answering over RDF Data.

[25] L Vilnis, X Li, S Murty, and A McCallum. 2018. Probabilistic Embedding of Knowledge Graphs with Box Lattice Measures. In *ACL*.

[26] Luke Vilnis and Andrew McCallum. 2015. Word Representations via Gaussian Embedding. arXiv:1412.6623 [cs.CL]

[27] M. Wang, H. Shen, S. Wang, L. Yao, Y. Jiang, G. Qi, and Y. Chen. 2019. Learning to Hash for Efficient Search Over Incomplete Knowledge Graphs. In *2019 IEEE International Conference on Data Mining (ICDM)*. 1360–1365.

[28] W Yih, X He, and C Meek. [n.d.]. Semantic Parsing for Single-Relation Question Answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems 30*. 3391–3401.

[30] W Zheng, J Yu, L Zou, and H Cheng. 2018. Question Answering over Knowledge Graphs: Question Understanding via Template Decomposition. (2018).

[31] W Zheng, L Zou, X Lian, J Yu, S Song, and D Zhao. 2015. How to Build Templates for RDF Question/Answering: An Uncertain Graph Similarity Join Approach. In *Proceedings of the 2015 ACM SIGMOD*.

APPENDIX: REPRODUCIBILITY

Reproducibility. Three datasets are used in our experiments, including FB15k, FB15k-237 and NELL. All experiments are performed on a machine with an Intel Core-i7 3.00GHz CPU, 64GB memory and Nvidia GeForce GTX 2080. The details of datasets, machine and parameters can be found in Section 4. All datasets are publicly available. The source code can be found at https://github.com/lihuiliullh/NewLook.

Subgraph Matching Baselines. GFinder is an indexing based subgraph matching algorithm which can find both exact and approximate subgraphs at the same time. FilM is a filtering based subgraph matching algorithm. G-Ray is an approximate attributed subgraph matching algorithm. Since G-Ray does not support edge type, for a fair comparison, we have extended G-Ray to support edge type [17], and we constrain its Neighbor-Expander to only select new nodes near the seed nodes.

A – Data Generation and Statistics

We use the same protocol as [19] to generate the training, validation and testing datasets. That is, we first divide all the edges (i.e., relations) R in the knowledge graph into three relation sets, including R_{train}, R_{valid} and R_{test}, where $R_{train} \subset R_{valid} \subset R_{test} = R$ and R_{train} contains 10% missing edges. The induced subgraph of these three relation sets are referred to as $\mathcal{G}_{train}$, $\mathcal{G}_{valid}$ and $\mathcal{G}_{test}$ (note that $\mathcal{G}_{test} = \mathcal{G}$). Given a DAG query q and a knowledge graph $\mathcal{G} \in \{\mathcal{G}_{train}, \mathcal{G}_{valid}, \mathcal{G}_{test}\}$, we use subgraph matching algorithm to find multiple subgraphs in $\mathcal{G}$, and merge all the subgraphs which share the same anchor nodes. Then starting from these anchor nodes, we execute the subgraph matching process and find all the answers for each of the variable nodes in the query.

For each query graph q, we denote A_{train}, A_{valid} and A_{test} as the answer sets obtained on $\mathcal{G}_{train}$, $\mathcal{G}_{valid}$ and $\mathcal{G}_{test}$, respectively. At the training time, we treat all the entities in A_{train} as the positive examples and other entities as the negative examples. At the validation and test phrases, we use the same strategy. The dataset details can be found at Table 4. The average number of answer entities for each query graph is shown in Table 5.

Table 4: Summary of datasets

Dataset	Entities	Relations	Training Edges	Validation Edges	Test Edges	Total Edges
FB15k	14,951	1,345	483,142	50,000	59,071	592,213
FB15k-237	14,505	237	272,115	17,526	20,438	310,079
NELL	63,361	200	114,213	14,324	14,267	142,804

Table 5: Average number of answer entities of test queries

Dataset	1p	2p	3p	2i	3i	ip	pi	2u	up	2d	dc
FB15k	10.8	255.6	250.0	90.3	64.1	593.8	190.1	27.8	227.0	3.3	59.5
FB15k-237	13.3	131.4	215.3	69.0	48.9	593.8	257.7	35.6	127.7	3.4	63.3
NELL	8.5	56.6	65.3	30.3	15.9	310.0	144.9	14.4	62.5	2.0	41.8

B – Additional Algorithm Details

Distance function. The point-to-box distance function we use in this paper is defined as follows [10].

$$d(v;\mathbf{b}_j) = d_o(v;\mathbf{b}_j) + \alpha \times d_i(v;\mathbf{b}_j),$$
$$d_o(v;\mathbf{b}_j) = ||\text{Max}(\mathbf{e}_v - q_{max}, 0) + \text{Max}(q_{min} - \mathbf{e}_v, 0)||_1 \quad (7)$$
$$d_i(v;\mathbf{b}_j) = ||\mathbf{b}_j^c - \text{Min}(q_{max}, \text{Max}(q_{min}, \mathbf{e}_v))||_1$$

where $q_{max} = \mathbf{b}_j^c + \mathbf{b}_j^o$ and $q_{min} = \mathbf{b}_j^c - \mathbf{b}_j^o$ and $0 < \alpha < 1$ is a fixed scalar. By this definition, the distance of a node (i.e., a point in the embedding space) to the box center is equal to the distance inside the box d_i and the distance outside the box d_o. α is a weight parameter to measure the importance of these two distances.

Table 6: Accuracy of each variable node. Q2B refers to Query2Box, NLK refers to NewLook, ✗ means the node does not exist in the corresponding query structure.

Method	GQE	Q2B	GRay	FilM	GFinder	NLK	GQE	Q2B	GRay	FilM	GFinder	NLK
Node	Node3						Node4					
2ipp	0.624	0.859	**0.942**	0.454	0.629	0.865	0.530	0.292	0.369	0.622	0.631	**0.663**
2ippu	0.668	0.857	**0.943**	0.469	0.633	0.897	0.478	0.253	0.392	0.640	0.593	**0.651**
2ippd	0.640	0.878	**0.927**	0.521	0.707	0.901	0.490	0.333	0.381	**0.689**	0.711	0.669
Node	Node5						Node6					
2ipp	0.497	0.292	0.352	0.624	**0.656**	0.632	✗	✗	✗	✗	✗	✗
2ippu	0.574	0.285	0.357	0.642	0.636	**0.676**	0.651	0.313	0.330	✗	0.665	**0.821**
2ippd	0.463	0.323	0.342	0.703	**0.681**	0.637	0.258	0.206	0.161	✗	**0.509**	0.357
Node	Node4						Node5					
3ipp	0.538	0.870	**0.883**	0.270	0.460	0.824	0.386	0.224	0.334	0.447	0.404	**0.597**
3ippu	0.581	0.876	**0.888**	0.282	0.454	0.869	0.412	0.218	0.328	0.448	0.452	**0.605**
3ippd	0.621	**0.903**	0.865	0.303	0.508	0.878	0.420	0.213	0.329	0.481	0.470	**0.581**
Node	Node6						Node7					
3ipp	0.419	0.233	0.323	0.465	0.448	**0.644**	✗	✗	✗	✗	✗	✗
3ippu	0.451	0.271	0.324	0.496	0.484	**0.661**	0.587	0.306	0.286	✗	0.471	**0.797**
3ippd	0.486	0.285	0.314	0.487	0.537	**0.684**	0.263	0.181	0.177	✗	**0.413**	0.395

Table 7: Accuracy comparison. gf refers to GFinder.

Method	gf	gf+NLK	gf	gf+NLK	gf	gf+NLK	gf	gf+NLK	gf	gf+NLK
Node	Node3		Node4		Node5		Node6		Time(s)	
Query1	0.629	0.579	0.631	0.584	0.656	0.585	✗	✗	111	44
Query2	0.633	0.583	0.593	0.496	0.636	0.567	0.665	0.543	153	47
Query3	0.707	0.671	0.711	0.641	0.681	0.595	0.509	0.408	145	63
Node	Node4		Node5		Node6		Node7		Time(s)	
Query4	0.460	0.424	0.404	0.362	0.448	0.460	✗	✗	124	44
Query5	0.454	0.390	0.452	0.314	0.484	0.411	0.471	0.373	149	58
Query6	0.508	0.478	0.470	0.403	0.537	0.491	0.413	0.337	163	66

Table 8: MRR results (%) on the Query2Box datasets.

Dataset	1p	2p	3p	2i	3i	ip	pi	2u	up	Average	Training Time
FB15k											
BetaE	65.0	42.1	37.8	52.9	64.0	41.5	22.9	48.8	26.9	44.6	2.81 days
EmQL	36.8	45.2	40.9	57.4	60.9	**55.6**	53.8	7.4	**37.5**	43.9	$\geq$ 4 days
NewLook	**84.1**	**56.6**	**45.1**	**60.1**	**77.9**	28.7	**56.9**	**82.5**	34.0	58.4	0.91 days
FB15k-237											
BetaE	39.1	24.2	20.4	28.1	39.2	19.4	10.6	22.0	17.0	24.4	2.33 days
EmQL	33.4	30.5	**30.4**	37.8	43.6	**35.1**	35.8	7.5	**24.1**	30.9	$\geq$ 4 days
NewLook	**77.8**	**39.6**	28.1	**55.2**	**64.5**	14.1	**36.0**	**61.6**	23.4	44.5	0.75 days
NELL											
BetaE	53.0	27.5	28.1	32.9	45.1	21.8	10.4	38.6	19.6	30.7	2.62 days
EmQL	37.2	35.1	34.9	53.9	65.4	**44.1**	**56.1**	10.5	31.1	40.9	$\geq$ 4 days
NewLook	**87.5**	**54.6**	**43.4**	**68.9**	**74.8**	19.7	42.2	**77.7**	**31.4**	55.6	0.87 days

C – Additional Experimental Results

Additional Experiments for BetaE vs. EmQL vs. NewLook. Table 8 shows the MRR of different methods. As we can see, NewLook has the shortest training time. Most of the time, it has the highest MRR.

Additional Experiments for Queries with Multi-variable Nodes. Table 3 shows the average accuracy for queries with multi-variable nodes. Table 6 further shows the accuracy of each variable node in a given query structures 2ipp, 2ippu, 2ippd, 3ipp, 3ippu and 3ippd. As we can see, NewLook has either the highest or the competitive accuracy in most cases. For node3 in 2ipp, 2ippu and 2ippd, and node4 in 3ippu and 3ippd, G-Ray has the highest accuracy. This is mainly because node3 and node4 are treated as seed nodes in G-Ray. On average, NewLook obtains more than 10% higher accuracy than other baseline methods. For some nodes (e.g. node5 and node6 in

3ipp and 3ippu), NewLook obtains more than 15% higher accuracy than the best competitor.

The Pruning Power of NewLook. For most subgraph matching methods, their online query time usually increases dramatically (e.g., quadratically or even exponentially) w.r.t the data graph size. A natural question is "*How can we prune some unpromising entities of the data graph, so that the size of data graph can be significantly reduced to accelerate the online query process?*" To answer this question, we conduct experiments to evaluate the pruning power of NewLook.

In the experiments, we use the query structures in Figure 8b. For each query, we use NewLook to find the top-50 candidates for each variable node, and put these candidates into a node set $\mathcal{F}$. Then, we find an induced data graph according to $\mathcal{F}$. In this way, the size of the data graph is significantly reduced. To evaluate the pros and cons of pruning, we run GFinder on the induced data graph, and compare its accuracy and query time before and after the pruning. Table 7 shows the results. As we can see, by running GFinder on the induced data graph, the online query time is reduced significantly (by about two-third), at the expense of a lightly decreased accuracy (about 5% to 10% on average).

D – Ablation Study and Error Analysis

First, we evaluate the effect of the nonlinear modeling of projection operation. Table 9 shows the ablation study of projection operation. As we can see from the first four columns of the table, the proposed neural network based projection operation has a much better performance than the linear transformation based projection operation, especially for multi-hop queries (e.g. 2p, 3p). For one-hop query (1p), the linear transformation (LT) performs slightly better than the neural network method (NN) in some cases. This is consistent with our intuition in that the proposed nonlinear projection method is most effective in mitigating cascading errors in multi-hop queries (e.g., 2p, 3p). The more accurate modeling of the projection operations also leads to better intersection operations, as shown in the last two columns of Table 9.

Second, we evaluate the effect of the proposed difference operation, where we use an attention network to capture the asymmetric property of the difference operation. We compare the proposed attention neural network based model with deepset [29] model. The results are shown in Table 10. As we can see, the proposed attention neural network based model has a much better performance than deepset based model. For query graph dp, the deepset model has a 0 hits score. It indicates that the generalization ability of the deepset model for modeling the difference operation is very limited. However, the proposed attention neural network based model still obtains a reasonable hits score.

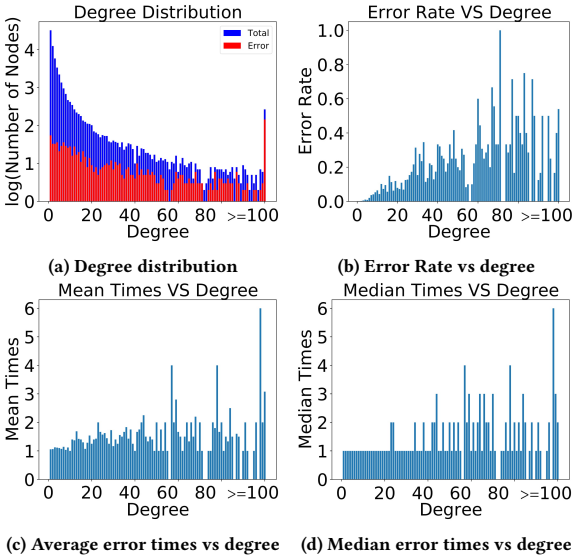
Third, we analyze where the error of the proposed NewLook comes from. We run the query 2ippu on NELL dataset, collect all the wrongly predicted nodes in H@10 and put them in a node list S_w. Figure 11a shows the degree distribution of the entire data graph and that of the nodes in S_w. The x-axis is degree and the y-axis is the number of nodes. Figure 11b shows the error rate of nodes with different degrees. If $x = 80$ and $y = 0.4$, this means that 40% nodes of degree 80 in the data graph returned by NewLook are wrong answers (i.e., they are contained in S_w). Figure 11c shows the average times a node is returned by NewLook as a wrong answer w.r.t. the node degree. If $x = 80$ and $y = 3$, this means that nodes in

Table 9: Ablation study of projection operation. LT refers to linear transformation based model. NN refers to the proposed neural network based model.

Query	1p		2p		3p		2i		3i	
Method	LT	NN	LT	NN	LT	NN	LT	NN	LT	NN
FB15k										
Hits@3	**0.71**	0.69	0.37	**0.41**	0.27	**0.38**	0.59	**0.66**	0.71	**0.78**
Hits@10	**0.86**	0.84	0.53	**0.57**	0.42	**0.53**	0.77	**0.80**	0.86	**0.89**
FB15k-237										
Hits@3	**0.43**	**0.43**	0.22	**0.26**	0.17	**0.23**	0.31	**0.34**	0.43	**0.47**
Hits@10	**0.60**	0.59	0.36	**0.39**	0.29	**0.35**	0.49	**0.51**	0.61	**0.63**
NELL										
Hits@3	0.53	**0.62**	0.25	**0.33**	0.22	**0.33**	0.28	**0.34**	0.46	**0.53**
Hits@10	0.70	**0.73**	0.39	**0.47**	0.35	**0.45**	0.48	**0.49**	0.66	**0.68**

Table 10: Ablation study of difference operation.

Query	2d		3d		dp	
Method	Deepset	Attention	Deepset	Attention	Deepset	Attention
FB15k						
Hits@3	0.67	**0.72**	0.52	**0.58**	0.00	**0.26**
Hits@10	0.82	**0.86**	0.67	**0.73**	0.00	**0.40**
FB15k-237						
Hits@3	0.56	**0.59**	0.43	**0.48**	0.00	**0.21**
Hits@10	0.73	**0.75**	0.60	**0.66**	0.00	**0.35**
NELL						
Hits@3	0.74	**0.75**	0.53	**0.59**	0.00	**0.29**
Hits@10	0.83	**0.85**	0.69	**0.74**	0.00	**0.43**



(a) Degree distribution **(b) Error Rate vs degree**

(c) Average error times vs degree **(d) Median error times vs degree**

Figure 11: Degree distribution and error rate distribution.

S_w with a degree of 80 are returned as wrong answers three times on average. Figure 11d shows the median time a node is returned as a wrong answer w.r.t. the node degree. As we can see, compared with low degree nodes, high degree nodes have a higher error rate. This indicates that NewLook prefers to choosing high degree nodes even though they might not be the true answers. We suspect this might be a general limitation of embedding based methods, since a higher degree node is more likely to be located in the proximity of an arbitrary box (e.g., a query) in the embedding space. A possible remedy is to leverage the node degree to 'regularize' the embedding process so as to mitigate the negative impact of high-degree nodes.