A Holistic Approach for Answering Logical Queries on Knowledge Graphs

Yuhan Wu
East China Normal University
yuhanwu0001@163.com

Yuanyuan Xu
Zhejiang Lab, University of New South Wales
xyycn8@163.com

Xuemin Lin
Shanghai Jiao Tong University
lxue@cse.unsw.edu.au

Wenjie Zhang
University of New South Wales
wenjie.zhang@unsw.edu.au

***Abstract*—Logical queries on Knowledge Graphs (KGs) is a fundamental sub-task of knowledge graph reasoning. A promising paradigm for answering logical queries, recently, has been proposed based on versatile deep learning techniques. In this line, the query is first broken down into a series of first-order logical predicates, and then both the query and knowledge graph entities are jointly encoded in the same embedding space. Some approaches are able to support the full range of traditional First-Order Logic (FOL) operations for complex queries in real-world scenarios, while others have attempted to create a new combination of FOL operations by replacing the negation operation with the difference operation due to the poor performance of the negation operation. Our empirical observations show that the difference operator is more effective for multi-hop reasoning, while the negation operator is better suited for use as the final operation in the query, particularly in single-hop settings. In addition, other fundamental limitations such as linear transformation assumption for negation operator and the fixed-lossy problem for difference operator further degrade the performance of these methods. In light of these, we propose the HaLk, a holistic approach for answering logical queries that, to our knowledge, is the first to support a full set of logical operators in a unified end-to-end framework. In this approach, we propose specific neural models for each operator by considering their own intrinsic properties, based on which HaLk effectively mitigates the cascading error of projection and negation operators as well as delicately provides closed-formed solutions for difference operator. Extensive experimental results on three datasets demonstrate that HaLk outperforms all competitors and achieves up to 32% improvement in accuracy.**

***Index Terms*—Logical queries, knowledge graph, geometry deep learning.**

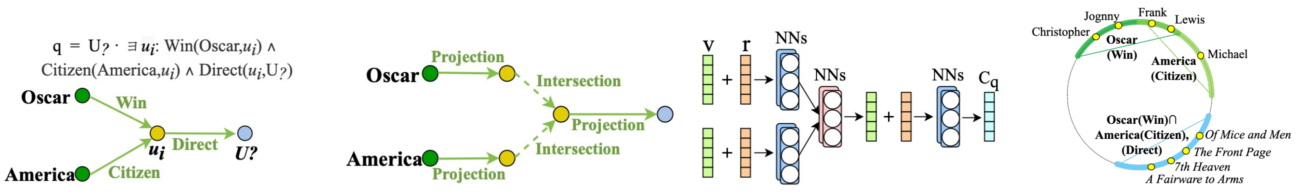
I. Introduction

Knowledge graphs are prevalent in real-life scenarios due to their ability to provide flexible, structured representation for entities and the intricate connections among them. Answering logical queries on knowledge graphs is a fundamental and important task that has a range of applications, including search engines, semantic webs [1], and recommendation systems [2], etc. The goal of this task is to identify answer entities in the knowledge graph that are likely to be entailed by known facts or inferred through generalization based on observed knowledge facts. In practical applications, logical queries on KGs require dealing with many kinds of complex logical operators over multi-hop relations. As shown in Fig. 1a and Fig. 1b, a natural language question like "*What are the films directed by Oscar-winning American directors?*" can be defined as the composition of logical predicates. Based on it, the aforementioned question can be first translated into a logical query graph and a corresponding computation graph after that. Next, query processing methods are proposed to provide the solutions, and desirable entities such as "*7th Heaven*" will be returned as the answers of the above query in Fig. 1d. Existing query processing methods can be roughly divided into subgraph matching-based methods and embedding-based ones.

The former one [3]–[5] relies on exact or approximate matching between the query graph and candidate subgraphs in KGs, which can be problematic due to incomplete or noisy KGs and may lead to empty or incorrect answers. To address these issues, an alternative solution is to use advanced deep learning techniques to embed the queries and KG entities in a vector space. These embedding-based methods [2], [6]–[9] are robust to handle incomplete or noisy KGs and have good response efficiency. Embedding-based methods can be further divided into three categories based on their support for first-order logical operations.

The first group [2], [10], [11] only considers existential positive first-order (EPFO) logical operations, thereby limiting their practicality. In the second group, negation operation is modelled based on the *linear* transformation assumption [8], [9], [12], which enables them to process negative logical queries. However, they suffer from the following problems: (1) The candidate answer set of the negation operator tends to be large owing to the nature of complementary set. For example, in Fig. 2(b), a general survey question "*Find out people who have never studied abroad.*" can be transformed into a form with the negation operator $\neg B$, whose answer entities may be larger than 1000 or even 10000. In this case, the linear one-to-multiple projection model for the negation operator does not perform well. This is why the accuracy of queries with the negation operator is often quite low (generally less than

Yuhan Wu and Yuanyuan Xu are the joint first authors.
Xuemin Lin is the corresponding author.



(a) Logical query graph (b) Computation graph (c) Embedding generation (d) Vector space

Fig. 1: Overall query processing procedures for a query *"What are the films directed by Oscar-winning American directors?"*.

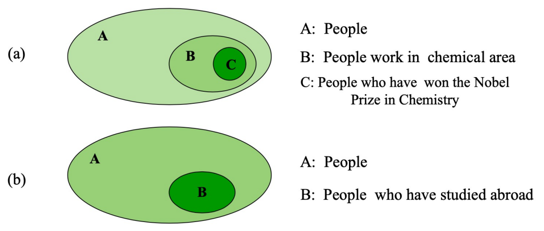

(a)
A
B
C
A: People
B: People work in chemical area
C: People who have won the Nobel Prize in Chemistry
(b)
A
B
A: People
B: People who have studied abroad

Fig. 2: Illustration of difference and negation operators.

20%), making it impractical for real-world applications. (2) Furthermore, for multi-hop queries with intermediate negation operators, the uncertainty of a large candidate answer set will disturb the overall learning process.

To better handle complex queries, some methods in the third group [6], [7] introduce an alternative operation called the difference operation to replace the negation operation, which avoids the problem of uncertainty. For example, consider the query "*Please find out the chemical researchers who have not won the Nobel Prize in chemistry.*". As shown in Fig. 2(a), it is better to translate this question into a logical query with the difference operator (i.e., B-C) rather than using the negation operator (i.e., $\neg$C $\wedge$ B), as has been empirically validated. However, the difference operator is not able to process one-hop queries like the one in Fig. 2(b) since it cannot define and model the *universal set* due to the limitations of these methods. More importantly, [6] does not offer an explicit method for modeling the difference operation; another one [7] is only able to model a partial answer region due to the limitations of box embedding, leading to false negatives/positives. When processing multi-hop queries, these errors can be amplified and accumulated, causing further degradation in model performance.

Our observation. The difference and negation operators have their own strengths. We empirically observe that difference operation performs better when processing multi-hop queries, as it generates compact but accurate candidate answer sets, while negation operation is more effective as the tail operation, particularly in single-hop settings. While these two operations complement each other, they cannot be perfectly interchangeable. However, none of the existing solutions is able to formulate and learn the five types of first-order logic operations in a unified framework, and they are incompatible with each other due to different embedding backbones.

To fill the gap, we propose a Holistic Approach to answer Logical queries on Knowledge graphs based on a new arc embedding (namely HaLk) in this paper, in which we introduce a full set of first-order logical operations, which is a union of traditional FOL operations and newly-defined FOL operations, covering projection, intersection, difference, negation and union. Specifically, we reformulate the difference operation and propose a new model with a closed-form solution [1] by taking semantic center, arc overlap, and cardinality constraint into account, which boosts the long and complex logical query processing. The negation operator is learned using neural networks to better model the dependency mapping between queries and multiple answer entities, and to reduce the cascading error from previous sub-queries, making it more effective and applicable. Similarly, HaLk optimizes the projection operation by giving more flexibility to the start and end points of the arc embedding based on their intrinsic properties to improve accuracy. Finally, HaLk provides a powerful pruning method for subgraph matching-based algorithms and significantly reduces online response time with only a slight sacrifice in accuracy.

To sum up, the main contributions of this paper are:

- To the best of our knowledge, this is the first attempt to introduce and support a new full set of first-order logical operations in a unified end-to-end framework, which enables HaLk more practical in real-world scenarios.
- We reformulate the difference operation and propose a new model based on its intrinsic characteristics, which can provide closed-formed solutions and facilitate the process of complex logical queries.
- We go beyond *linear* transformation assumption and model logical operations (i.e., projection and negation) by considering their properties to mitigate the cascading error, thus significantly improving the accuracy.
- Extensive experimental results demonstrate that HaLk achieves up to averaged 32% improvement over the best competitor on three benchmark datasets in MRR and Hit@3 metrics; meanwhile, HaLk is significantly faster than subgraph matching-based methods.

[1]The solution of a logical operator can be expressed in the closed form if the region of the answer set is still the arc segment, which is slightly different from the definition of mathematical expression.

II. Preliminaries

In this section, we introduce basic concepts, give the problem definition and then discuss some related works.

A. Basic Concepts

Knowledge Graphs. A knowledge graph is denoted as $\mathcal{G} = \{V, R, T\}$ where $V = \{v_1, v_2, \cdots, v_n\}$ is the entity set, $R = \{r_1, r_2, \cdots, r_m\}$ is the relation set and T is the fact triplet set. Each triplet in KGs can be denoted as (h, r, t) where $h, t \in V$ is a subject (head entity) and an object (tail entity), respectively, and $r \in R$ is the predicate (relation) of the triplet that connects h to t. Suppose that $a_j(\cdot, \cdot) \in \mathcal{A}$ is a binary function $a_j : \mathcal{V} \times \mathcal{V} \to \{1, 0\}$ corresponding to r_j, where $a_j(h_i, t_k) = 1$ if and only if (h_i, r_j, t_k) is a factual triple. Note that $\mathcal{A}$ is the relational function set. To avoid learning from scratch, we randomly divide all the nodes in KGs into different groups with video memory-friendly size and record the group ownership of each node by one-hot vectors. In addition, a relation-based 3D adjacency matrix is adopted to track the connectivity between groups based on each predicate. Specifically, we use a row one-hot vector to denote the group attribute of each node v_i. If node v_i belongs to group j then $[\boldsymbol{h}_i]_j = 1$, otherwise, it equals to 0. As for the adjacency matrix, for each $r_j \in R$, $\mathcal{M}_j^{ik} = 1$ if any node in group i connects with any node in group k by relation r_j, else $\mathcal{M}_j^{ik} = 0$.

First-Order Logic (FOL). Following the definition in [2], a FOL query contains an anchor entity set $\tilde{U} \subset V$ and a variable node set. Here, the variable node set includes existentially quantified bound variables $u_1, u_2, \cdots, u_k$ and the target variable $u_?$ (i.e., the answers to the query). Then, a FOL query q is defined as [2]

$$q[u_?] = u_? . \exists\ u_1, u_2, \cdots, u_k : \tau_1 \vee \tau_2 \vee \cdots \vee \tau_n, \tag{1}$$

where $\exists$ is existential quantifier operator, $\vee$ is logical disjunction operator, and τ_i are logical conjunctions sub-items, i.e., $\tau_i = \varpi_{i1} \wedge \cdots \wedge \varpi_{im}$, $\wedge$ is the logical conjunction operator. Here $\varpi_{ij} = a(\tilde{u}_a, u_i)$ or $\neg a(\tilde{u}_a, u_i)$ or $a(u_i, u_j)$ or $a(u_i, u_?)$, and $\neg$ is the logical negation operator. Using the above notations, answering a query q is equivalent to *finding a set of entities* $[\![q]\!] \subset V$, where $v \in [\![q]\!]$ if and only if $q[v]$ is True.

Computation Graphs. We represent a logical query as a Directed Acyclic computation Graph (DAG): $\mathcal{Q} = \{U, R, L\}$, where $U = \{\tilde{U}, U_?\}$ refers to the node set with $\tilde{U}$ denotes the anchor nodes that are the source nodes of DAG and $U_?$ denotes the variable nodes, R denotes the relation set that is the same as the relation set of $\mathcal{G}$, and L denotes logical operations. We give an example of logical query graph and computation graph in Fig. 1a and Fig. 1b. When building the computation graph for the query, it has been shown through literature [2], [12] and our own experiments that the order of operator selection should be $projection > intersection/difference > negation > union$ for effectiveness.

[2]We use FOL queries in its Disjunctive Normal Form (DNF) [2], which represents FOL queries as a disjunction of conjunctions.

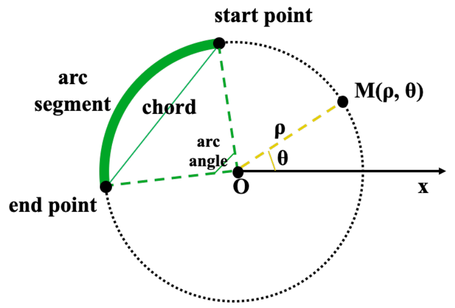


Fig. 3: The illustration of polar coordinate system.

Arc Embedding. In this paper, we propose the arc embedding as the basis for HaLk based on the rotation paradigm, in which embedding based on the rotation paradigm has been applied to many tasks [8], [13]–[16]. Given a KG $\mathcal{G}$ and a logical query graph $\mathcal{Q}$, we aim to embed each entity of the KG as a *point* on the circle with radius ρ, namely *point embedding* and encode each node of the query graph as an arc segment on the same circle, namely *arc embedding*, in polar coordinates, as shown in Fig. 3. We elaborate the arc embedding below.

For the arc embedding, we aim to represent the semantic center with a center embedding and the cardinality of the answer set with an arclength embedding. Therefore, we model the answer region of $[\![q]\!]$ as a Cartesian product of the *arc* with a fixed radius in all dimensions. Specifically, we denote the semantic center point and the arc span/cardinality of $[\![q]\!]$ with the parameter A_c and the parameter A_l, respectively. If the embedding dimensionality is d, we use a d-ary Cartesian product to define the arc segment as $\boldsymbol{A}_q = ((A_c^1, A_l^1), \cdots, (A_c^d, A_l^d))$, where $A_c^i = (\rho, A_{c,a}^i)$, $A_{c,a}^i$ are polar angles of the center points, $A_{c,a}^i \in [0, 2\pi)^d$. $A_l^i \in [0, 2\pi\rho]^d$ are arclengths. For simplicity, we also omit radius learning and will study it in future work. Thus $\boldsymbol{A}_c$ refers to the polar angle in this paper.

For an entity/node $v \in V$, it can be regarded as an entity set with a single element, i.e., $\{v\}$. Thus, we can represent an entity as a Cartesian product of arc segments with arclengths $\mathbf{0}$, where the center point indicates the entity semantic. In Fig. 1d, each yellow dot on the circle denotes an entity embedding, and each green or blue arc segment on the same circle denotes a query arc embedding in the embedding space.

Logical Operations. To our knowledge, we are the first to introduce a full set of logical operations, which is a union of traditional FOL operations and newly-defined FOL operations, containing projection $\mathbb{P}$, difference $\mathbb{D}$, intersection $\mathbb{I}$, negation $\mathbb{N}$ and union $\mathbb{U}$. Fig. 4 illustrates the five types of logical operations in the embedding space.

B. Problem Statement

Based on the above notation, the key challenge of answering logical queries on knowledge graphs is to effectively generate the arc embedding for the target node in $\mathcal{Q}$, the point embedding for each entity in $\mathcal{G}$, and the arc embedding for each relation $r \in R$. Here, we focus on the single target variable and formally define the problem as follows.

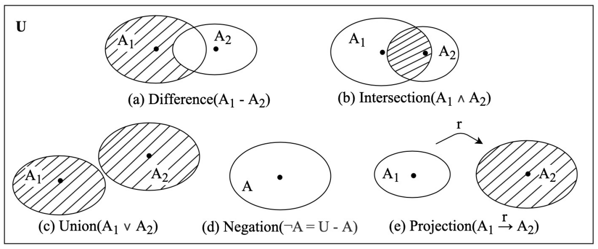


Fig. 4: Illustration of five logical operations. The shaded region is the desirable output of the logical operation.

Problem 1: Given a knowledge graph $\mathcal{G} = \{V, R, T\}$, a logical query graph $\mathcal{Q}$ with anchor node(s) and variable nodes, the problem of answering logical queries on knowledge graphs aims to find a set of entities as answers such that these entity embeddings should be included or close to the arc embedding of the given query in the embedding space.

C. Related Work

We review related studies in two categories: subgraph matching-based methods and embedding-based methods.

Subgraph matching based methods first represent logical queries as DAGs and then obtain a set of answers via subgraph matching methods [3]–[5]. While simple and intuitive, such approaches have many drawbacks including long response time. In addition, subgraph matching is very sensitive as it has difficulty in correctly answering queries with missing relations and noisy information [17], [18]. One potential solution is to impute missing relations, but this will lead to a denser knowledge graph, which would exacerbate the issue [19].

Another category is embedding based works, which embed logical queries and entities of KGs into the same embedding space. They can robustly handle missing relations while being faster than subgraph matching based methods due to the simple vector computation when answering online queries. In this line, existing solutions learn query embeddings and entity embeddings based on different embedding backbones including geometric shapes [2], [10], [20], probability distributions [12], and other complex objects [6], [21]. We divide them into three groups from the perspectives of logical operations.

The first group only support existential positive first-order (EPFO) logical operations [2] or a subset of EPFO logical operations [10], [20]. However, they are not capable of processing complex logical queries with negative operations, thereby limiting their practicability. To cope with such drawbacks, some works in the second group manage to support four types of logical operations (including EPFO logical operations and negation). Specifically, BetaE [12] proposed a probabilistic embedding framework for answering arbitrary FOL queries over KGs; ConE [8] learnt logical operations based on cone embedding and MLPMix [9] employed the MLP to model logical queries. Among them, *linear* transformation assumption was employed to model negation operation, which limits the fitting ability of the negation operator for the one-to-multiple relations, and might generate a biased answer area with high uncertainty in the embedding space. Meanwhile, the intrinsic characteristics of negation operator may lead to dramatically poor performance, especially for multi-hop queries.

Alternatively, researchers attempt to define a new operation to replace the negation operation, based on which four types of logical operations (including EPFO logical operations and difference operator) can be supported in the last group. NewLook [7] was able to support difference operator, but its answer region cannot be exactly represented by a hyper-rectangle box embedding, and it would introduce false negative entities or false positive entities in the answer set, and even cascading errors with the iteration of multi-hop reasoning. EmQL [6] claimed that it can support difference operation but did not provide the concrete algorithm in the original paper. In addition, these methods are not able to process single-hop logical queries with negation operation since the *universal set* cannot be defined in these methods.

To summarize, HaLk has several key advantages over existing works. (1) We propose the arc embedding paradigm that allows us to derive closed-form solutions for all operators, which is not possible in existing approaches. (2) We introduce a coordinated information pair consisting of a start point and an end point, which can work together to adjust both location and range information, reducing cascading errors. (3) HaLk supports five logical operations, making it more versatile and applicable in a wider range of situations compared to existiing methods which support a maximum of four logical operators.

III. Our Approach

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

A. Overall Framework

We aim to map the logical query graphs (q) and the knowledge graph (G) into a low-dimensional space, where each q is mapped as arc segments and entities of G are represented as points on the same circle. Entities could be answers to the query are included or close to the arc segments of the query target node. During the training stage, the computation graph is regarded as a sequence of geometric logical operations and each logical operation is modelled using a neural network (see Fig. 1c). We then learn entity embeddings and relation embeddings for $\mathcal{G}$, as well as the parameterized neural networks for all the logical operators, until the convergence of the model.

For the online step, the trained model generates the arc embedding for the target node of the test logical query by executing a logical operation set. Using the obtained result arc embedding, we estimate the probability that the knowledge graph entities satisfy the query using a search algorithm in the low-dimensional vector space, such as a nearest neighbor search. In the answer identification phase, we retrieve entities inside or close to the target arc embedding as candidate answers to the query.

In the following subsections, we will introduce specific implementations of five logical operations and provide details on model training.

B. Projection Operation

The projection operator $\mathbb{P}$ typically transforms one entity set into another one using the given relation (see Fig. 4(e)). This operator is used to answer many basic questions, such as "*Who are authors of <Deep Learning>?*". Many previous works [2], [10], [20] suffer from severe cascading error [22] in modelling projection operator due to the *linear* transformation assumption. Here, cascading error arises from two parts: the center embedding deviates from the ideal semantic center and the range embedding does not match the set cardinality. Later, some solutions [7], [8] attempt to employ non-linear neural networks to alleviate the cascading error problem. However, they independently learn the center and cardinality of the answer region, which leads to a semantic gap between the center embedding and range embedding. Furthermore, simply concatenating both as input to the learning model lacks guidance for their combination and association, so the learning process of the semantic center and set cardinality cannot benefit from each other to alleviate cascading error.

To bridge the semantic gap, we introduce a combination representation, including the start and end point, which includes both center and cardinality information. This allows for rotating and scaling to be performed in a cooperative manner. we first give the definitions below.

Definition 1 (Start point of an arc segment.): Given an arc embedding $\boldsymbol{A} = (\boldsymbol{A}_c, \boldsymbol{A}_l)$, the start point is $\boldsymbol{A}_c - \boldsymbol{A}_l/(2\rho)$, denoted as $\boldsymbol{A}_S$.

Definition 2 (End point of an arc segment.): Given an arc embedding $\boldsymbol{A} = (\boldsymbol{A}_c, \boldsymbol{A}_l)$, the end point is $\boldsymbol{A}_c + \boldsymbol{A}_l/(2\rho)$, denoted as $\boldsymbol{A}_E$.

Based on Definitions 1 and 2, we can adaptively fit the candidate answer region by directly adjusting the start point and end point. This process employs both center and cardinality information to generate the target arc embedding, which helps to bridge the "semantic gap" caused by the independent learning of center and cardinality. Later we will describe how we use them to model each operator.

Given the head arc embedding $\boldsymbol{A}_h$, we first obtain an approximate arc embedding $(\tilde{\boldsymbol{A}}_c, \tilde{\boldsymbol{A}}_l)$ by rotations of head embedding according to the relation embedding $\boldsymbol{r} = (\boldsymbol{A}_{r,c}, \boldsymbol{A}_{r,l})$, that is, $\tilde{\boldsymbol{A}}_c = \boldsymbol{A}_{h,c} + \boldsymbol{A}_{r,c}, \tilde{\boldsymbol{A}}_l = \boldsymbol{A}_{h,l} + \boldsymbol{A}_{r,l}$. Based on Definitions 1 and 2, we compute the start point $\tilde{\boldsymbol{A}}_S$ and end point $\tilde{\boldsymbol{A}}_E$, and take them as the input of the learning network. We formulate the learning model of the projection operator as

$$\begin{aligned} &\boldsymbol{A}_c = g(\mathrm{MLP}(\tilde{\boldsymbol{A}}_S \| \tilde{\boldsymbol{A}}_E)) \quad \boldsymbol{A}_\alpha = g(\mathrm{MLP}(\tilde{\boldsymbol{A}}_S \| \tilde{\boldsymbol{A}}_E)), \\ &\boldsymbol{A}_l = \rho \boldsymbol{A}_\alpha. \end{aligned} \tag{2}$$

Here, MLP is a multi-layer perceptron [23], and $\|$ represents the concatenation of two vectors. $\boldsymbol{A_\alpha}$ is the corresponding arc angles of the arclengths. Since center embedding represents angles and arclength embedding models arclengths, it is necessary to overcome the information gap between them with the help of arc angles. $g(\cdot)$ is a function that regulates $\boldsymbol{A}_c \in [0, 2\pi)^d$ and $\boldsymbol{A}_\alpha \in [0, 2\pi]^d$. We define $g(\cdot)$ as

$$[g(\boldsymbol{x})]_i = \pi \tanh(\lambda x_i) + \pi. \tag{3}$$

where $[g(\boldsymbol{x})]_i$ denotes the i-th element of $g(\boldsymbol{x})$, λ is a fixed parameter to control the scale. The center embedding $\boldsymbol{A}_c$ and arclength embedding $\boldsymbol{A}_l$ of the answering region are generated by using neural networks to learn the combination representations jointly in Eq. (2) and (3). Through the collaborative adjustment of rotation and scaling transformations, the output would cover corresponding answer entities more accurately, thus cascading errors are greatly reduced.

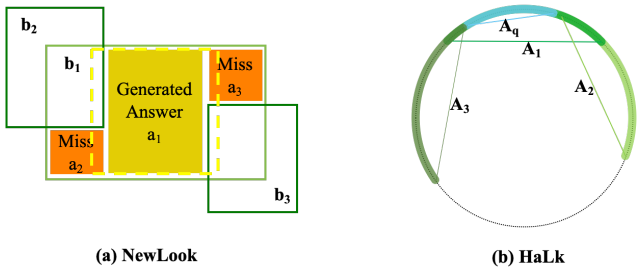


Fig. 5: Illustration of two models for difference operator.

C. Difference Operation

Difference operation is a useful logical operation as shown in Fig. 4(a). It aims to answer questions like "*Who won the world championships in the badminton but didn't win the Olympic Games?*".

To date, only NewLook [7], based on box embedding, can provide solutions with the inevitable lossy problem for the modelling of difference operator. However, NewLook often struggles with the difference operation due to its inability to produce a precise valid box in most cases where input box embeddings have partial overlaps. This can lead to either false positives/negatives in the answer region. For instance, in Fig. 5(a), the ideal result of the answer region for $\boldsymbol{b}_1 - \boldsymbol{b}_2 - \boldsymbol{b}_3$ should include both the yellow and orange regions. If the learning process focuses on including more correct answer entities, it may also result in false positive entities being included in the answer region. On the other hand, if the model is trained to minimize the number of wrong answers, it may result in false negatives.

Furthermore, although the attention neural network is proved to be effective in modelling the difference operator based on box embedding in NewLook, the usual raw-value attention calculation is not compatible with the rotation-based embedding backbone due to the periodicity of the rotation transformation. The ordinary weighted average may result in inconsistent semantics, which is illustrated in Supplementary [24].

To avoid the fixed-lossy problem and semantic inconsistency, we aim to reformulate the difference operation to provide a closure solution. The challenges are summarized below: (1) The difference operation is *asymmetric* to the input order. For example, the result of input $\{\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_k\}$ should be different from the result of input $\{\boldsymbol{A}_2, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_k\}$, with the former being inside $\boldsymbol{A}_1$ and the latter being inside $\boldsymbol{A}_2$. When the original input is $\{\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_k\}$, no matter how the order of $\boldsymbol{A}_2, \cdots, \boldsymbol{A}_k$ is changed, the result should be invariant to permutations. However, off-the-shelf permutation-invariant neural networks cannot directly model the asymmetry of part of the original input. (2) How to quantify the overlap of multiple arcs while taking the periodicity into account since the polar angle of center embedding is periodic. (3) How to leverage the semantic center scheme and cardinality constraint in modelling? Due to the fact that the result $[\![q]\!]$ is always the subset of the input $[\![q_1]\!]$, $\boldsymbol{A}_c$ should be inside $\boldsymbol{A}_{1,c}$ and $\boldsymbol{A}_l$ should be no longer than $\boldsymbol{A}_{1,l}$. To address them, we propose a new model to learn the difference operator, which will be introduced from the semantic center view and the arclength view, respectively.

Semantic average centers. Specifically, to avoid the periodic problem and learn the semantic average centers, we first convert the coordinates of input center points to rectangular coordinates by

$$\boldsymbol{A}_{i,c,r} = (\boldsymbol{x}_i,\ \boldsymbol{y}_i) = (\rho\cos(\boldsymbol{A}_{i,c}),\ \rho\sin(\boldsymbol{A}_{i,c})) \tag{4}$$

where $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is the corresponding rectangular coordinates of center points embedding $\boldsymbol{A}_{i,c}$. $\cos(\cdot)$ and $\sin(\cdot)$ are element-wise cosine and sine functions. Then, we model the arc embedding $\boldsymbol{A}_c$ using an attention mechanism-based network in the rectangular coordinate system. Formally, the computation process is

$$\begin{gathered}\boldsymbol{x}_{sa} = \sum_{i=1}^{k} \boldsymbol{w}_i \odot \boldsymbol{x}_i \qquad \boldsymbol{y}_{sa} = \sum_{i=1}^{k} \boldsymbol{w}_i \odot \boldsymbol{y}_i \\ \boldsymbol{A}_{sa} = (\rho,\ \arctan(\boldsymbol{y}_{sa}/\boldsymbol{x}_{sa})), (\boldsymbol{x}_{sa} \neq 0)\end{gathered} \tag{5}$$

Here, $(\boldsymbol{x}_{sa},\ \boldsymbol{y}_{sa})$ is the generated semantic average center points in rectangular coordinates and $\boldsymbol{w}_i$ are positive weight vectors that satisfy $\sum_{i=1}^{k}[\boldsymbol{w}_i]_j = 1$ for all $j = 1, \cdots, d$. The computation of $\boldsymbol{w}$ is given in Eq. (7). After obtaining the semantic average center points, we restore the coordinates to the polar coordinate system $\boldsymbol{A}_{sa}$. Then, the result arc embedding $\boldsymbol{A}_c$ is obtained by

$$\begin{aligned}&\boldsymbol{A}_c = \text{Reg}(\boldsymbol{x}_{sa}, \boldsymbol{y}_{sa}),\\ &\text{Reg}([\boldsymbol{x}_{sa}]_j, [\boldsymbol{y}_{sa}]_j) = \begin{cases} (\boldsymbol{A}_{sa})_j + \pi, & if\ [\boldsymbol{x}]_j < 0, [\boldsymbol{y}]_j > 0 \\ (\boldsymbol{A}_{sa})_j - \pi, & if\ [\boldsymbol{x}]_j < 0, [\boldsymbol{y}]_j < 0 \\ \boldsymbol{A}_{sa}, & otherwise. \end{cases}\end{aligned} \tag{6}$$

Here, $j = 1, 2, \cdots, d$. $\text{Reg}(\cdot)$ [3] is the function that regularize the value range of the final result to be in a single period.

To compute weights $\boldsymbol{w}_i$ in Eq. (5), we take the start point and end point of the arc segment ($\boldsymbol{A}_{i,S}$ and $\boldsymbol{A}_{i,E}$) as weight measurement factors to better measure the cross-correlation among arc segments:

$$\boldsymbol{w}_i = \frac{\exp(\boldsymbol{\kappa}_i \text{MLP}([\boldsymbol{A}_{i,S} \| \boldsymbol{A}_{i,E}]))}{\sum_{j=1}^{k}\exp(\boldsymbol{\kappa}_j \text{MLP}([\boldsymbol{A}_{j,S} \| \boldsymbol{A}_{j,E}]))}, \tag{7}$$

where $i = 1, 2, \cdots, k$ $\boldsymbol{\kappa}_i$ is a weighted vector to make $\boldsymbol{A}_c$ inside the $\boldsymbol{A}_1$. Meanwhile, $\boldsymbol{\kappa}$ can also consider the different effects of rest input entity sets on the final result independently of the input order.

[3]Note that we manually set $[\boldsymbol{x}]_j$ to be a small number (e.g., 10^{-3}) when $[\boldsymbol{x}]_j = 0$, to avoid the illegal division.

Arclengths with cardinality constraint. To calculate the arclengths, we consider the cardinality constraint, periodicity, and overlap computation. Specifically, we hard code the asymmetry between $\boldsymbol{A}_1$ and $\boldsymbol{A}_i$ ($i = 2, \cdots, k$) by taking the overlaps between $\boldsymbol{A}_1$ and $\boldsymbol{A}_2, \cdots, \boldsymbol{A}_k$ as the initial input formation of model fitting. Hence, we compute $\boldsymbol{A}_l$ of $\boldsymbol{A}_q$ by

$$\boldsymbol{A}_l = \boldsymbol{A}_{1,l} \cdot \sigma(\text{DeepSets}(\{\boldsymbol{A}_1 - \boldsymbol{A}_j\}_{j=2}^{k})), \tag{8}$$

where $\sigma(\cdot)$ is the element-wise sigmoid function, $\text{DeepSets}(\cdot)$ is a permutation-invariant function [25], and $\{\boldsymbol{A}_1 - \boldsymbol{A}_j\}_{j=2}^{k}$ satisfies the permutation invariance among $\boldsymbol{A}_j$. When evaluating the overlap, there is a information gap between angle and length, and the polar angles of center points are periodic. To address these issues, we use the chord lengths of $\boldsymbol{A}_{1,c}$ and $\boldsymbol{A}_{i,c}$ to measure the degree of overlap between center points. $\text{DeepSets}(\{\boldsymbol{A}_1 - \boldsymbol{A}_j\}_{j=2}^{k})$ is then computed by

$$\begin{aligned}&\text{MLP}(\frac{1}{k-1}\sum_{j=2}^{k}\text{MLP}([\boldsymbol{\delta}_c \| \boldsymbol{\delta}_l]))\\ &\boldsymbol{\delta}_c = 2 \cdot \rho \cdot \sin((\boldsymbol{A}_{1,c} - \boldsymbol{A}_{j,c})/2), \quad \boldsymbol{\delta}_l = \boldsymbol{A}_{1,l} - \boldsymbol{A}_{j,l}\end{aligned} \tag{9}$$

Last, the boundary of answer region returned by HaLk is tighter than the one returned by NewLook, which is theoretically analysed in the Supplementary.

D. Intersection Operation

Intersection operation is a widely-used logic operation (see Fig. 4(b)). It is applied to answer questions like *"which animals can live on the land and under the water?"*.

Similar to the difference operation, we use the semantic center scheme to model the intersection operation to avoid semantic inconsistency. Additionally, we utilize the coarse-grained random group information to guide the learning process. Specifically, we compute the one-hot/multi-hot vector for U_t by $\boldsymbol{h}_{Ut} = \boldsymbol{h}_{U1} \odot \boldsymbol{h}_{U2} \odot \cdots \odot \boldsymbol{h}_{Uk}$, where $\odot$ is the element-wise product. We expect that arc segments with similar group information should be highly correlated. For instance, if $\boldsymbol{h}_{Ui}$ is similar to $\boldsymbol{h}_{Ut}$, arc segment $\boldsymbol{A}_i$ should accordingly have a greater impact on the intersected output. We provide more details on how we use semantic centers and arclengths to implement this technique below.

Semantic average centers. Like with the difference operator, we use attention neural networks in the rectangular coordinate system to learn the semantic center embedding $\boldsymbol{A}_c$ using the same initial computation process as in Eq. (4), (5) and (6). Moreover, since the intersection operator is permutation invariant to all the input order, we can use the similarity of group information and the similarity of the (start point embedding, end point embedding) pair representation together as the weight measurement factor to measure the cross-correlation among input arc segments:

$$\boldsymbol{w}_i = \frac{\exp(\boldsymbol{z}_i \text{MLP}([\boldsymbol{A}_{i,S} \| \boldsymbol{A}_{i,E}]))}{\sum_{j=1}^{k}\exp(\boldsymbol{z}_j \text{MLP}([\boldsymbol{A}_{j,S} \| \boldsymbol{A}_{j,E}]))}, \tag{10}$$

where $i = 1, 2, \cdots, k$ and $\boldsymbol{A}_{i,S}$, $\boldsymbol{A}_{i,E}$ are the start point and end point of $\boldsymbol{A}_i$. $\boldsymbol{z}_i = 1/(\|\boldsymbol{h}_{Ui} - \boldsymbol{h}_{Ut}\| + 1)$ reflects the similarity between the two entity sets. Using Eq. (10), due to the different degrees of cross-correlation with target arc segment, we can learn different effects that the k input arc embeddings should have on the results, so as to locate the semantic center of the candidate answer set more accurately.

Arclengths with cardinality constraint. Since $[\![q]\!]$ is the subset of all $[\![q_i]\!]$ $(i = 1, 2, \cdots, k)$, $\boldsymbol{A}_l$ should be no larger than any input arclengths $\boldsymbol{A}_{i,l}$. To identify the appropriate arc segment for the target, we incorporate cardinality constraint and center location information into the modeling process and also consider the arc angles of the arclengths as intermediate results. The computation process is

$$\begin{aligned} &\boldsymbol{A}_{i,\alpha} = \boldsymbol{A}_{i,l}/\rho \quad for\ i = 1, 2, \cdots, k, \\ &\boldsymbol{A}_{\alpha} = \min\{\boldsymbol{A}_{1,\alpha}, \cdots, \boldsymbol{A}_{k,\alpha}\} \cdot \sigma([\text{DeepSets}(\{\boldsymbol{A}_j\}_{j=1}^{k})]), \end{aligned} \tag{11}$$

Here, $\min\{\boldsymbol{A}_{1,\alpha}, \cdots, \boldsymbol{A}_{k,\alpha}\}$ ensures that the cardinality of the output entity set is no larger than the minimum one of input, and the DeepSets$(\cdot)$ network is used to obtain a permutation invariant comprehensive influence factor. Specifically, given the input of start point embeddings and end point embeddings, DeepSets$(\{\boldsymbol{A}_j\}_{j=1}^{k})$ is defined as

$$\text{MLP}(\text{Mean}(\sum_{j=1}^{k} \text{MLP}([\boldsymbol{A}_{j,S} \| \boldsymbol{A}_{j,E}]))), \tag{12}$$

where Mean$(\cdot)$ is the dimension-wise mean function. Last, we transfer the arc angle embeddings $\boldsymbol{A}_\alpha$ into arclength embeddings $\boldsymbol{A}_l$ by $\boldsymbol{A}_l = \rho \boldsymbol{A}_\alpha$.

E. Negation Operation

Negation operator is used for negative queries. From the modelling and performance perspective, it is more suitable to be the tail operation of logical queries, especially when dealing with single-hop problems like the universality survey problem.

Only a few methods (i.e., BetaE, ConE, and MLPMix) support the negation operation. However, a major disadvantage is that they all model the negation operator based on the assumption of *linear* transformation, which limits their inference ability since *linear* transformation restricts the model's ability to fit complex mappings between queries and their large answer entity set. In this paper, we use a non-linear neural network to learn negation operation, which helps fit the dependency mapping between queries and their multiple answer entities as well as mitigate the cascading errors accumulated from previous sub-queries. Moreover, to represent the semantic difference between $[\![q]\!]$ and $[\![\neg q]\!]$, we assume that the included angle between their center points to be π, and the arc segment of $[\![q]\!]$ and $[\![\neg q]\!]$ should form a complete circle. We first obtain an approximate arc embedding $\tilde{\boldsymbol{A}}_{\neg q} = (\tilde{\boldsymbol{A}}_{\neg c}, \tilde{\boldsymbol{A}}_{\neg l})$ to help determine the initial transformation direction via linear transformation as follows:

$$\begin{aligned} &[\tilde{\boldsymbol{A}}_{\neg c}]_i = \begin{cases} [\boldsymbol{A}_c]_i + \pi, & if\ [\boldsymbol{A}_c]_i \in [0, \pi) \\ [\boldsymbol{A}_c]_i - \pi, & if\ [\boldsymbol{A}_c]_i \in [\pi, 2\pi) \end{cases} \\ &[\tilde{\boldsymbol{A}}_{\neg l}]_i = 2\pi\rho - [\boldsymbol{A}_l]_i \qquad [\tilde{\boldsymbol{A}}_{\neg\alpha}]_i = [\tilde{\boldsymbol{A}}_{\neg l}]_i/\rho, \end{aligned} \tag{13}$$

where $i = 1, 2, \cdots, d$, $\tilde{\boldsymbol{A}}_{\neg\alpha}$ is the corresponding arc angle of $\tilde{\boldsymbol{A}}_{\neg l}$. Then, we treat $\tilde{\boldsymbol{A}}_{\neg c}$ and $\tilde{\boldsymbol{A}}_{\neg\alpha}$ as the input of the neural network to obtain the final centers and arclengths for the query target node. We define the neural network below.

$$\begin{aligned} &\boldsymbol{t}_1 = \text{MLP}(\tilde{\boldsymbol{A}}_{\neg c}) && \boldsymbol{t}_2 = \text{MLP}(\tilde{\boldsymbol{A}}_{\neg\alpha}) \\ &\boldsymbol{A}_{\neg c} = g(\text{MLP}(\boldsymbol{t}_1 \| \boldsymbol{t}_2)) && \boldsymbol{A}_{\neg\alpha} = g(\text{MLP}(\boldsymbol{t}_1 \| \boldsymbol{t}_2)). \\ &\boldsymbol{A}_{\neg l} = \rho \boldsymbol{A}_{\neg\alpha} \end{aligned} \tag{14}$$

Here, $g(\cdot)$ is defined in Eq. (3). By Eq. (14), we first obtain the intermediate results $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ using neural networks. We then learn the center embedding and arc angle embedding by jointly using $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, which allows us to better fit the answer entity set. In cases where the negation operation is the final operation in the input query, non-linear transformation can also be used to correct errors that may have accumulated from sub-queries in previous iterations. Finally, we obtain the arclength embedding by transforming the target arc angle embedding.

F. Union Operation

Unlike the previous logical operators, we aim to provide an *exact* solution for the union operation, so we do not model it using a neural network. In some cases, the k input arc segments may not form single result region, as shown in Fig. 4(c), thus the output of the union operator may not be a single arc segment. If we model the result of union as a single arc segment, it would include a large false positives. To solve this issue, we adopt the DNF technique [2], which allows us to transform *any* union operation to the last step of the computation graph.

Specifically, for the computation graph $Q = (U_q, R_q)$ of a given FOL query q, let U_{union} be the set of nodes whose prepositive operators are "union", and let P_u be the set of parent nodes for each $u \in U_{union}$. DNF will generate $N = \prod_{u \in U_{union}} |P_u|$ new computation graphs. And each computation graph corresponds to a conjunctive query. After that, we only need to find the answers for each of these computation graphs and take the union of the answers as the result of union operation. Therefore, the union operator for HaLk is non-parametric and corresponds to the *exact* set. In practical applications, both the number of union operations and the inputs for each union operator in a query are typically very small (i.e., ≤ 5). Additionally, the N conjunctive sub-queries can also be executed in parallel, so the additional cost caused by the DNF technique is completely acceptable.

G. Model Training

To obtain the answers of a given query based on the principle of maximum similarity, we expect that the point embeddings of entities $v \in [\![q]\!]$ are pushed closer to the arc embedding of q, and the point embeddings of entities $v' \notin [\![q]\!]$ are pulled away from the arc embedding of q. We achieve this by defining a function to measure the distance between a given query embedding and an entity embedding in KGs. Besides, we use a negative sampling trick to help train the model efficiently, where m negative samples are selected from the negative entity set using the random sampling strategy.

Distance Function. First, we define the function to measure the distance between entities and the query. Inspired by [2], the distance d comprises two parts: the outside distance d_o and the inside distance d_i. Suppose that $\boldsymbol{v} = (\boldsymbol{A}_{v,c}, \boldsymbol{0})$, $\boldsymbol{A}_q = (\boldsymbol{A}_c, \boldsymbol{A}_l)$, $\boldsymbol{A}_S = \boldsymbol{A}_c - \boldsymbol{A}_l/2\rho$ and $\boldsymbol{A}_E = \boldsymbol{A}_c + \boldsymbol{A}_l/2\rho$. We define the distance function as

$$d(\boldsymbol{v}\|\boldsymbol{A}_q) = d_o(\boldsymbol{v}\|\boldsymbol{A}_q) + \eta d_i(\boldsymbol{v}\|\boldsymbol{A}_q), \tag{15}$$

where $0 < \eta < 1$ is a weighted parameter to down-weight the within-arc distance. Furthermore, we consider the periodicity problem and take the corresponding chord length of the included angle that won't lead to duality due to the periodicity as the measurement of the distance between the two points. Hence, the outside distance and the inside distance are respectively calculated by

$$\begin{aligned} d_o &= 2\rho\|\min\{|\sin((\boldsymbol{A}_{v,c} - \boldsymbol{A}_S)/2)|, |\sin((\boldsymbol{A}_{v,c} - \boldsymbol{A}_E)/2)|\}\|_1 \\ d_i &= 2\rho\|\min\{|\sin((\boldsymbol{A}_{v,c} - \boldsymbol{A}_c)/2)|, |\sin((\boldsymbol{A}_l/2\rho)/2)|\}\|_1. \end{aligned} \tag{16}$$

where $\|\cdot\|_1$ is the ℓ_1 norm, $\sin(\cdot)$ and $\min(\cdot)$ are element-wise sine and minimization functions. The parameter $\eta \in (0, 1)$ is fixed during training, so that $\boldsymbol{v}$ is encouraged to be inside the arc $\boldsymbol{A}_q$, but not necessarily be equal to the centers of $\boldsymbol{A}_q$ (i.e., semantic center point). Since the union operator is represented by a set of arc embeddings, the distance function d defined above is not directly applicable. Therefore, we take the minimum distance between the entity point embedding and the set of query arc embeddings to be the result as the Disjunctive Normal Form [2].

Loss Function. Given a set of logical queries during the training stage, we optimize the following loss function

$$\begin{aligned} Loss = &-\log\sigma(\gamma - d(\boldsymbol{v}\|\boldsymbol{A}_q) - \xi\|\text{Relu}(\boldsymbol{h}_v - \boldsymbol{h}_{U_q})\|_1) \\ &- \frac{1}{m}\sum_{i=1}^{m}\log\sigma(\xi\|\text{Relu}(\boldsymbol{h}_{v_i'} - \boldsymbol{h}_{U_q})\|_1 + d(\boldsymbol{v}_i'\|\boldsymbol{A}_q) - \gamma), \end{aligned} \tag{17}$$

where $\gamma > 0$ is a fixed margin, $v \in [\![q]\!]$ is a positive entity, $v_i' \notin [\![q]\!]$ is the i-th negative entity, m is the number of negative entities and $\sigma(\cdot)$ is the sigmoid function. The term $\xi\|\text{Relu}(\boldsymbol{h}_v - \boldsymbol{h}_{U_q})\|_1$ is used to measure the distance between the one-hot vector $\boldsymbol{h}_v$ and one-hot/multi-hot vector $\boldsymbol{h}_{U_q}$. In summary, the overall training procedure of the proposed HaLk is summarized in Algorithm 1.

It is worth mentioning that, our framework and ConE are both based on the rotation paradigm, but there are major differences: (1) Based on the proposed arc embedding with chord length as the new distance measurement standard, we avoid the duality of results caused by the periodicity of the angle in ConE. (2) HaLk has the potential to capture hierarchical relations in KGs through the polar radius, while ConE cannot. (3) The issue of cascading error is further addressed by introducing the start point embedding and end point embedding, which bridge the semantic gap between center modeling and cardinality modeling. (4) HaLk improves the applicability of logical queries in practice by implementing five operators in a unified model.

H. Complexity Analysis

First, we give the complexity of each operator during the training stage followed by the online cost. The time complexity of training stage includes five parts as follows. For the projection/negation operator, the time complexity is $\mathcal{O}(|B|d)$, where $|B|$ denotes the batch size and d represents the dimension of query embeddings and entity embeddings. The complexity of difference/intersection operation is $\mathcal{O}(|B|kd + |B|d)$, where k denotes the number of input entities (e.g., $3i, k = 3$). Union operation takes $\mathcal{O}(1)$ cost since it only needs to simply gather the results of n conjunctive queries.

During the online stage, answering a logical query simply involves processing the n conjunctive queries in Eq. (1), where n is typically small in practice. Note that all n computations can be parallelized. Additionally, we execute a sequence of simple logical operations based on arc embedding for each conjunctive query. The time required for each conjunctive query is a simple superposition of the execution times of the operators it contains, which is usually a small constant time. To get the final answers, we perform a range search in the low-dimensional vector space, which can also be done in constant time using search algorithms such as Locality Sensitive Hashing (LSH) [26]. In total, processing a logical query online is very fast, as discussed in Section IV-E. Plus, the online time will not significantly increase as the knowledge graphs/queries gradually expand.

IV. Experiments

In this section, we give the experimental configurations and conduct comprehensive experiments to evaluate the performance of the proposed HaLk.

A. Experimental Settings

For a fair comparison, we follow the commonly-used experimental configurations in [7]–[9].

Datasets. We evaluate our approach on three standard knowledge graphs: FB15k [20], FB15k-237(FB237) [27], and NELL995 (NELL) [28]. We create three graphs respectively for training, validation and test, which satisfies $\mathcal{G}_{training} \subseteq \mathcal{G}_{validation} \subseteq \mathcal{G}_{test}$.

Baselines. We compare HaLk against three state-of-the-art methods, including ConE [8], NewLook [7], and MLPMix [9], all of which are published in the last two years. NewLook is SOTA among methods that can support newly-defined FOL operations, ConE and MLPMix are SOTA among methods that can support traditional FOL operations. Note that ConE and MLPMix do not support difference operator, and NewLook do not support negation operator. Thus corresponding results are not included in the experiments.

Queries. We collect the same query structures from the baselines, whose query workloads are in their public repository. We obtain the 16 basic query structures in total, including 12 query structures (that is, 1p, 2p, 3p, 2i, 3i, ip, pi, 2u, up, 2d, 3d, dp) from NewLook, and 4 query structures with negation (that is, 2in, 3in, pin, pni) from ConE and MLPMix, as shown in Fig. 4 in the Supplementary. To assess the generalization

Algorithm 1: HaLk Algorithm

```
Input: Queries $Q$ with positive entity $V_p$ and negative
       entities $V_n$, knowledge graph triples;
Output: Entity embeddings, relation embeddings;
1  Initialize $e \in E$ and $r \in R$ randomly;
2  for epoch$\leq$ MaxEpoch or !convergence do
3      for batch query with the same structure $\in Q$ do
4          loss = 0;
5          while ($op = nextOp(\cdot) \neq NULL$) do
6              if op == 'projection' then
7                  Compute the arc embedding $(\boldsymbol{A}_c, \boldsymbol{A}_l)$
                     by Eq. (2);
8              if op == 'intersection' then
9                  Compute the arc embedding $(\boldsymbol{A}_c, \boldsymbol{A}_l)$
                     by Eq. (10), (11), (12);
10             if op == 'difference' then
11                 Compute the arc embedding $(\boldsymbol{A}_c, \boldsymbol{A}_l)$
                     by Eq. (6), (7), (8), (9);
12             if op == 'negation' then
13                 Compute the arc embedding $(\boldsymbol{A}_c, \boldsymbol{A}_l)$
                     by Eq. (13), (14);
14             if op == 'union' then
15                 Union of $k$ input query embeddings;
16         Compute the loss according to Eq. (17) ;
17         epoch+=1;
18 return $E$, $R$
```

ability of the model, complex query structures, i.e., ip, pi, 2u, up and dp, are only evaluated in the validation and test stages.

Training protocol. We implement HaLk in Pytorch on four Nvidia RTX 3090 GPU with 1TB RAM. And we set embedding dimensionality $d = 800$, $\eta = 0.02$ (in Eq. (15)) and $\gamma = 24$ (in Eq. (17)). We use the uniform distribution to initialize the entity embeddings and relation embeddings. In each iteration, we sample a mini-batch of 512 queries and set the negative sampling number to 128. We optimize the loss function in Eq. (17) using Adam optimizer [29] with a learning rate of 0.0001.

Evaluation protocols. We adopt two evaluation protocols, i.e., Mean Reciprocal Rank (MRR) and Hits at K (Hit@K), both of which are used in baselines [7]–[9]. We average the evaluation scores over all the queries within the same query structure, and report the results separately for different query structures. For both evaluation metrics, a higher score indicates better performance.

B. Effectiveness of Query Processing

Following the same settings as the baseline methods, we compare HaLk against ConE, MLPMix and NewLook on queries with and without negation operation. We evaluate the effectiveness on three benchmark datasets under two widely-used evaluation metrics (i.e., MRR and Hit@3).

Queries without negation operation. We can observe from Tables I and II that (1) HaLk significantly and consistently outperforms three baselines across all the query structures, including those not seen (i.e., ip, pi, 2u, up, dp) during the training stage. Overall, we gain averaged 14% higher MRR and averaged 12% higher Hit@3 than the best competitor on FB15k dataset, averaged 15% higher MRR and Hit@3 than the best competitor on FB237 dataset, as well as averaged 16% higher MRR and Hit@3 than the best competitor on NELL dataset. (2) For multi-hop queries (e.g., 3p, ip, pi, dp), HaLk achieves up to $31\%, 29\%, 36\%$ absolute improvement over the best competitor on FB15k, FB237 and NELL, respectively. It suggests that closure modelling for the operations is beneficial for processing complex queries based on arc embedding backbone since the cascading errors can be alleviated effectively. In particular, the improvement of dp structure confirms that the difference operator is suitable for multi-hop queries due to the compact but accurate candidate answers. (3) Compared to ConE that is also rotation-based embedding backbone, the average accuracy of HaLk is 1.3-3 times that of ConE on three datasets in terms of MRR and Hit@3, which suggests the effectiveness of the proposed neural operators by considering their own intrinsic properties. (4) In contrast to MLPMix which employs the non-geometrical method to embed queries, HaLk consistently beats it by a substantial margin (i.e., up to 60% improvement in MRR and Hit@3); meanwhile, NewLook and ConE also perform better than MLPMix, which implies that geometry-based methods might be beneficial for logical queries on knowledge graphs. This may be because geometry-based methods have the ability to model and employ the cardinality of answer sets during the training stage. (5) For the queries with difference operation (i.e., 2d, 3d, dp), HaLk yields up to 8%, 5%, and 21% improvements in MRR and Hit@3 against NewLook among three benchmark datasets. It means that the proposed difference model can alleviate the effects of false negatives and false positives, demonstrating the superiority of modelling with closed-form solutions in HaLk. (6) We can observe that queries with projection operation universally outperform those of baselines, which indicates that the introduction of the start point and end point of the arc segment can boost each other and alleviate the cascading errors. (7) For new queries unseen during the training stage, HaLk gets an impressive improvement, which shows that HaLk generalizes well within and beyond query structures.

Queries with negation operation Tables III and IV show the results of HaLk against ConE and MLPMix on modelling FOL queries with negation. Overall, HaLk outperforms ConE and MLPMix by a large margin but the overall performance of all methods is on the low level. This is because the size of answer sets of queries is very large, sometimes up to 4000. Besides, it is observed that HaLk gains up to double improvements than baselines on NELL dataset in terms of MRR and Hit@3, which means that the neural model to negation operator can make the target arc segment to get close and cover more true answers. For the FB15k dataset, HaLk is slightly better than baselines in most cases, which suggests

TABLE I: MRR results (%) for answering queries on FB15k, FB237, and NELL. The best result is highlighted in bold.

Dataset	Method	Query												
		1p	2p	3p	2i	3i	ip	pi	2u	up	2d	3d	dp	Average
FB15k	ConE	73.1	34.0	29.2	64.4	73.4	35.6	51.0	55.3	31.6	-	-	-	49.7
	Newlook	84.1	56.6	45.1	60.1	77.9	28.7	56.9	82.5	34.0	92.0	45.8	42.8	58.9
	MLPMix	71.4	29.0	24.8	60.0	70.7	32.7	47.7	39.6	26.0	-	-	-	44.7
	Ours	**98.4**	**72.7**	**57.9**	**81.1**	**86.7**	**59.8**	**67.6**	**95.6**	**48.1**	**97.3**	**47.5**	**64.3**	**73.1**
FB237	ConE	42.1	12.7	11.0	32.4	47.6	14.8	26.2	14.1	10.0	-	-	-	23.4
	Newlook	79.7	42.3	29.3	62.6	70.7	22.2	39.7	64.4	23.3	86.8	36.1	37.7	49.6
	MLPMix	41.5	11.5	9.8	33.5	47.2	14.0	24.9	14.4	9.2	-	-	-	22.9
	Ours	**97.0**	**63.9**	**41.3**	**72.6**	**76.1**	**48.1**	**52.7**	**93.6**	**36.2**	**94.5**	**39.6**	**55.5**	**64.3**
NELL	ConE	53.2	16.1	14.0	39.9	50.4	17.5	26.3	15.4	11.3	-	-	-	27.1
	Newlook	86.2	57.0	45.6	72.0	79.1	26.0	44.0	76.7	31.3	91.5	44.5	53.8	59.0
	MLPMix	55.4	16.5	13.9	39.5	51.0	18.3	25.7	14.7	11.2	-	-	-	27.4
	Ours	**96.1**	**79.4**	**59.4**	**87.2**	**88.8**	**59.3**	**71.5**	**91.8**	**47.6**	**96.0**	**49.9**	**68.3**	**74.6**

TABLE II: Hit@3 results (%) for answering queries on FB15k, FB237, and NELL.

Dataset	Method	Query												
		1p	2p	3p	2i	3i	ip	pi	2u	up	2d	3d	dp	Average
FB15k	ConE	81.3	37.3	31.7	71.1	80.2	38.8	56.5	61.5	34.4	-	-	-	54.8
	Newlook	88.8	63.9	51.4	72.3	77.7	31.2	61.5	93.8	35.8	94.2	52.4	45.7	64.1
	MLPMix	79.1	32.0	27.0	66.4	77.0	35.8	52.7	45.1	28.1	-	-	-	49.2
	Ours	**99.8**	**76.0**	**60.8**	**84.5**	**89.4**	**61.9**	**71.3**	**99.1**	**51.8**	**97.1**	**55.3**	**63.8**	**75.9**
FB237	ConE	47.2	13.4	11.2	37.9	52.7	15.4	25.8	15.0	11.6	-	-	-	25.6
	Newlook	85.6	46.5	31.7	67.8	74.9	23.9	43.7	71.5	25.8	91.2	41.9	40.1	53.7
	MLPMix	45.8	11.7	9.8	37.5	52.2	14.5	27.2	15.1	9.2	-	-	-	24.8
	Ours	**99.4**	**68.9**	**45.5**	**76.6**	**79.2**	**51.4**	**56.2**	**98.0**	**39.8**	**96.8**	**46.2**	**60.5**	**68.2**
NELL	ConE	58.4	17.2	14.5	44.9	56.5	18.8	28.5	16.6	11.9	-	-	-	29.7
	Newlook	90.5	62.2	49.7	77.2	83.4	28.8	47.4	83.5	34.6	94.6	50.0	59.6	63.5
	MLPMix	58.8	15.4	13.6	43.7	55.0	15.9	26.8	14.0	10.8	-	-	-	28.2
	Ours	**98.6**	**85.3**	**64.7**	**90.0**	**92.1**	**64.1**	**75.3**	**95.9**	**52.5**	**98.4**	**56.6**	**74.2**	**79.0**

TABLE III: MRR results (%) for answering queries with negation on FB15K, FB237, and NELL.

Dataset	Method	Query				
		2in	3in	pni	pin	AVG
FB15k	ConE	18.0	18.6	15.2	9.7	15.4
	MLPMix	16.2	17.2	14.5	8.1	14.0
	Ours	**18.6**	**19.0**	**16.3**	**10.2**	**16.0**
FB237	ConE	5.7	9.4	3.9	4.4	5.9
	MLPMix	6.3	10.8	4.4	4.5	6.5
	Ours	**7.8**	**11.7**	**4.5**	**5.0**	**7.3**
NELL	ConE	5.5	7.8	3.9	3.9	5.3
	MLPMix	5.3	8.9	3.4	3.8	5.4
	Ours	**11.6**	**15.5**	**7.7**	**6.6**	**10.4**

TABLE IV: Hit@3 results (%) for answering queries with negation on FB15K, FB237, and NELL.

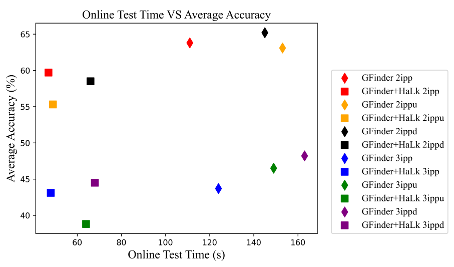
Dataset	Method	Query				
		2in	3in	pni	pin	AVG
FB15k	ConE	18.4	19.9	15.5	9.0	15.7
	MLPMix	16.8	18.3	14.7	7.1	14.2
	Ours	**19.8**	**21.4**	**16.1**	**10.3**	**16.9**
FB237	ConE	4.8	8.6	3.0	3.4	5.0
	MLPMix	5.6	10.1	3.4	3.6	5.7
	Ours	**7.8**	**12.1**	**4.4**	**4.4**	**7.2**
NELL	ConE	3.9	6.3	2.8	2.8	4.0
	MLPMix	4.0	7.9	2.5	2.7	4.3
	Ours	**12.0**	**16.7**	**7.5**	**6.2**	**10.6**

that we need to make further exploration when processing large answer sets and dense data graphs. There remains a large room to further improve and optimize the negation operator by considering its properties including a very large answer set and limited query instances for learning in knowledge graphs.

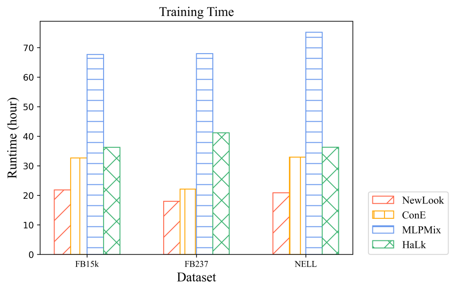
C. Ablation Study

We conduct the following ablation studies for difference, negation and projection operators on the NELL dataset under two evaluation metrics. All ablated networks are trained on the same experimental environment.

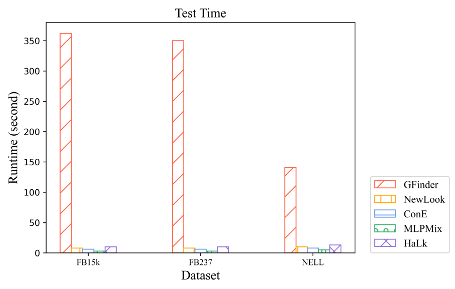
Difference operation. As we can see from the main effectiveness experiment above, closure modelling of HaLk is better



(a)



(b)



(c)

Fig. 6: (a) Accuracy and query time of GFinder before and after the pruning. (b) Offline time of different methods on three datasets. (c) Online time of different methods on three datasets.

TABLE V: Ablation study on NELL under MRR and Hit@3.

Difference	Hit@3			MRR		
	2d	3d	dp	2d	3d	dp
HaLk-V1	97.3	46.0	61.7	92.8	40.2	44.1
HaLk	**98.4**	**56.6**	**74.2**	**96.0**	**49.9**	**68.3**
Negation	Hit@3			MRR		
	2in	3in	pin	2in	3in	pin
HaLk-V2	9.7	12.5	4.5	9.5	11.7	5.0
HaLk	**12.0**	**16.7**	**6.2**	**11.5**	**15.5**	**6.6**
Projection	Hit@3			MRR		
	1p	2p	3p	1p	2p	3p
HaLk-V3	97.7	79.1	59.5	94.8	73.4	54.6
HaLk	**98.6**	**85.3**	**64.7**	**96.1**	**79.4**	**59.4**

than approximation modelling of NewLook. Here, we invoke the overlap computation of NewLook to replace that of HaLk and remove cardinality constraint to learn difference operator, dubbed HaLk-V1. It is observed from Table V that the proposed arclength computation in the HaLk achieves a much better performance than the HaLk-V1. For instance, HaLk can gain up to 14% improvement than HaLk-V1 on unseen query structures (i.e., dp), which indicates the superiority of the proposed difference model. Meanwhile, this also explains that cardinality constraint facilitates generating more appropriate regions for answers to logical queries.

Negation operation. We adapt the HaLk with the *linear* transformation to model the negation operator, namely HaLk-V2. We can see from Table V that HaLk with non-linear transformation is better than HaLk-V2 under both metrics. It means that HaLk has the ability to adaptively adjust the biased arc segments and mitigate the cascading error accumulated from previous paths. Besides, HaLk-V2 still outperforms ConE and MLPMix which also follow the *linear* transformation assumption, which suggests that HaLk can better learn projection and intersection operators with smaller errors.

Projection We integrate the projection model of NewLook, which performs better than the other two baselines, into HaLk, namely HaLk-V3. We can see from Table V that HaLk is more effective in alleviating cascading error, especially in multi-hop queries (e.g., 2p, 3p). This also suggests that the introduction of start point and end point of arc segment contributes to adjusting the biased arcs by simultaneously using the location information and range information. Furthermore, the more accurate modelling of the projection operator also leads to better intersection operations as shown in Tables I and II.

D. The Pruning Power of HaLk

To evaluate the pruning ability of HaLk, we perform experiments on HaLk to offer a candidate set for subgraph matching based methods, and we select the SOTA method GFinder [5] as the base. We take 6 query structures (i.e., 2ipp, 2ippu, 2ippd, 3ipp, 3ippu, 3ippd) as test structures to evaluate the pruning performance on NELL dataset in terms of accuracy. For each query, we use HaLk to obtain top-20 candidates for each variable node and add these candidates into a node set $\mathcal{S}$. After that, an induced data graph based on $\mathcal{S}$ could be generated, and we thus start from these anchor nodes to execute the GFinder on the induced data graph. Finally, we make a comparison of GFinder's accuracy and online query time before and after the pruning.

As observed from Fig. 6a, the pruning strategy provided by HaLk can significantly reduce the online query time of GFinder by approximately two-thirds with acceptable accuracy sacrifice (about 5% on average), which implies that HaLk can help speed up subgraph matching based methods as a pruning strategy. This also provides a promising direction to further optimize subgraph matching based methods by using the versatility of neural networks. For example, embedding based methods can become an alternative to the index structure of the data graph, which can reduce the large cost of index construction.

E. Efficiency of Query Processing

To evaluate the efficiency of our HaLk, we perform experiments on three datasets and report the offline training time and online query time. Here we select the same six query structures as the experiment in Section IV-D, each of which includes 100 queries. For the offline time, we calculate the

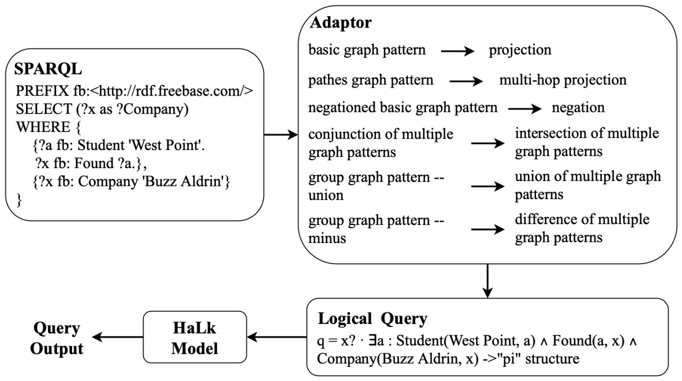

SPARQL
PREFIX fb:<http://rdf.freebase.com/>
SELECT (?x as ?Company)
WHERE {
{?a fb: Student 'West Point'.
?x fb: Found ?a.},
{?x fb: Company 'Buzz Aldrin'}
}
Adaptor
basic graph pattern → projection
pathes graph pattern → multi-hop projection
negationed basic graph pattern → negation
conjunction of multiple graph patterns → intersection of multiple graph patterns
group graph pattern -- union → union of multiple graph patterns
group graph pattern -- minus → difference of multiple graph patterns
Logical Query
q = x? · ∃a : Student(West Point, a) ∧ Found(a, x) ∧ Company(Buzz Aldrin, x) ->"pi" structure
HaLk Model
Query Output

Fig. 7: Procedure of answering a SPARQL query with the HaLk executor.

total training time while taking the average running time of these query structures. For the online query time, since the index in the GFinder method is built dynamically according to the characteristics of query, the time for building the index should be included in the test time.

We can observe from Fig. 6b that the non-geometry method MLPMix costs the most time during the training stage, while geometric methods (i.e., NewLook, ConE, and HaLk) take comparable computation costs. Note that HaLk accepts and learns a new full set of logical operations including five types of operators, while the other two methods only support four ones. Thus, HaLK takes slightly more training time than them.

On the other, embedding based methods are much faster than subgraph matching based methods in terms of online time since the time required for a query is only a simple superposition of the numerical calculation time of the operator networks it contains, regardless of the size of the knowledge graph. And the query times of embedding based methods are competitive as shown in Fig. 6c.

F. Application: SPARQL Query Answering

Our HaLk can be integrated into the broad landscape of query answering as the query executor and be compatible with real query languages. To demonstrate this, we take the widely-used query language SPARQL [30] as an example and give the detailed query procedure. Specifically, we provide a query $Adaptor$ to map between graph patterns and our five logical operators, as shown in Fig. 7(b), which also illustrates the importance of supporting more types of first-order logical operations to cover more scenarios in practice. Using the query $Adaptor$, we can first obtain the corresponding logical query for the given SPARQL query, and then HaLk can provide the query results. We visualize the process of answering a given SPARQL query on FB15k-237 test set using HaLk as the query executor in Fig. 7. The query results are shown in the Supplementary due to space limitation. Note that the process of generating query plans for graph patterns can occur through the query transformation and optimization layers in the query engine. For simplicity, these layers are omitted in Fig. 7.

TABLE VI: Accuracy and execution time of different sizes of queries on NELL. Here, 'QS': Query Size, 'EQS': Example Query Structure, 'H': HaLk, 'G': GFinder, 'ET': Execution Time.

QS	1		2		3		4		5	
EQS	1p		2p		pi		pip		p3ip	
Acc	H	G	H	G	H	G	H	G	H	G
	99.4	89.3	93.7	84.2	87.9	73.9	74.8	64.5	75.8	60.7
ET (ms)	H	G	H	G	H	G	H	G	H	G
	28	148	41	237	85	562	100	894	108	1205

G. Scalability with Query Size

To further evaluate the scalability of our HaLk, we conduct experiments on different sizes of queries compared to subgraph matching-based method (i.e., GFinder) on NELL dataset. As mentioned before, both HaLk and GFinder can serve as the query executor, where the inputs of query graphs can be generated by SPARQL queries using the Adaptor. We report the query execution time and accuracy in Table VI. The results indicate that HaLk performs better than GFinder in both accuracy and runtime, especially for larger query sizes. HaLk is approximately 12 times faster than GFinder and improves accuracy by up to 15.1% for large queries. HaLk's runtime slightly increases with an increase in query complexity while GFinder's runtime dramatically increases, which is consistent with our theoretical analysis before. This suggests that embedding-based methods like HaLk can significantly optimize the query engine as an effective and efficient query executor compared to subgraph matching-based methods. This presents an opportunity for future research to explore more advanced settings of the embedding-based query executor.

V. Conclusion

In this paper, we introduce a new full set of logical operations for the first time and propose a holistic approach to model them. We propose effective models for logical operations to mitigate the cascading error according to their own properties, thereby making the HaLk suitable for processing multi-hop, complex queries. HaLk is capable of providing elegant solutions for five operations in a closed form, while the existing solutions cannot. Extensive experimental results demonstrate that HaLk consistently outperforms baselines by a large margin (up to 32% improvement in MRR and Hit@3); meanwhile, HaLk can serve as a pruning method for subgraph matching based algorithms, which can significantly reduce online query time and provide a promising direction to optimize subgraph matching based methods by employing neural networks.

References

[1] M. Arenas, C. Gutiérrez, and J. F. Sequeda, "Querying in the age of graph databases and knowledge graphs," in *SIGMOD '21: International Conference on Management of Data*. ACM, 2021, pp. 2821–2828.

[2] H. Ren, W. Hu, and J. Leskovec, "Query2box: Reasoning over knowledge graphs in vector space using box embeddings," in *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[3] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, P. Berkhin, R. Caruana, and X. Wu, Eds. ACM, 2007, pp. 737–746.

[4] B. Du, S. Zhang, N. Cao, and H. Tong, "FIRST: fast interactive attributed subgraph matching," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1447–1456.

[5] L. Liu, B. Du, J. Xu, and H. Tong, "G-finder: Approximate attributed subgraph matching," in *2019 IEEE International Conference on Big Data*. IEEE, 2019, pp. 513–522.

[6] H. Sun, A. O. Arnold, T. Bedrax-Weiss, F. Pereira, and W. W. Cohen, "Faithful embeddings for knowledge base queries," in *Advances in Neural Information Processing Systems*, 2020.

[7] L. Liu, B. Du, H. Ji, C. Zhai, and H. Tong, "Neural-answering logical queries on knowledge graphs," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2021, pp. 1087–1097.

[8] Z. Zhang, J. Wang, J. Chen, S. Ji, and F. Wu, "Cone: Cone embeddings for multi-hop reasoning over knowledge graphs," in *Advances in Neural Information Processing Systems*, 2021, pp. 19 172–19 183.

[9] A. Amayuelas, S. Zhang, X. S. Rao, and C. Zhang, "Neural methods for logical reasoning over knowledge graphs," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=tgcAoUVHRIB

[10] W. L. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec, "Embedding logical queries on knowledge graphs," in *Advances in Neural Information Processing Systems*, 2018, pp. 2030–2041.

[11] N. Choudhary, N. Rao, S. Katariya, K. Subbian, and C. K. Reddy, "Self-supervised hyperboloid representations from logical queries over knowledge graphs," in *The Web Conference*. ACM / IW3C2, 2021, pp. 1373–1384.

[12] H. Ren and J. Leskovec, "Beta embeddings for multi-hop logical reasoning in knowledge graphs," in *Advances in Neural Information Processing Systems*, 2020.

[13] O. Ganea, G. Bécigneul, and T. Hofmann, "Hyperbolic entailment cones for learning hierarchical embeddings," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1646–1655.

[14] Z. Sun, Z. Deng, J. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019.

[15] Z. Zhang, J. Cai, Y. Zhang, and J. Wang, "Learning hierarchy-aware knowledge graph embeddings for link prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, 2020, pp. 3065–3072.

[16] Y. Bai, Z. Ying, H. Ren, and J. Leskovec, "Modeling heterogeneous hierarchies with relation-specific hyperbolic cones," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 12 316–12 327.

[17] L. D. Raedt, *Logical and relational learning*, ser. Cognitive Technologies. Springer, 2008.

[18] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proc. IEEE*, vol. 104, no. 1, pp. 11–33, 2016.

[19] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.

[20] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems*, 2013, pp. 2787–2795.

[21] D. Garg, S. Ikbal, S. K. Srivastava, H. Vishwakarma, H. P. Karanam, and L. V. Subramaniam, "Quantum embedding of knowledge for reasoning," in *Advances in Neural Information Processing Systems*, 2019, pp. 5595–5605.

[22] K. Guu, J. Miller, and P. Liang, "Traversing knowledge graphs in vector space," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. The Association for Computational Linguistics, 2015, pp. 318–327.

[23] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[24] [n.d], "Supplementary of "a holistic approach for answering logical queries on knowledge graphs"," https://github.com/yuhanwu0001/HaLk/tree/master.

[25] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, 2017, pp. 3391–3401.

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas*. ACM, 1998, pp. 604–613.

[27] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

[28] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2017, pp. 564–573.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2015.

[30] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gstore: answering sparql queries via subgraph matching," *Proceedings of the VLDB Endowment*, vol. 4, no. 8, pp. 482–493, 2011.