electronics

Article

UnTiCk: Unsupervised Type-Aware Complex Logical Queries Reasoning over Knowledge Graphs

Deyu Chen [1,2,3,4], Qiyuan Li [1,2,3,4] and Jinguang Gu [1,2,3,4,*]

1 Department of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
2 The Key Laboratory of Rich-Media Knowledge Organization and Service of Digital Publishing Content, Institute of Scientific and Technical Information of China, Beijing 100038, China
3 Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial System, Wuhan 430065, China
4 Big Data Science and Engineering Research Institute, Wuhan University of Science and Technology, Wuhan 430065, China
* Correspondence: simon@wust.edu.cn

Abstract: For the task of answering complex logical queries on large-scale incomplete knowledge graphs, the promising approach is to embed the knowledge graph and complex logical queries into a low-dimensional space and perform iterative reasoning to find the final answers. The general problem is that these models do not include entity types as an important feature, which reduces the reasoning potential. However, explicit type information is not always available on large-scale knowledge graphs. We innovatively propose an embedding-based framework for Unsupervised Type-Aware Complex Logical Queries (UnTiCk). Our approach implements unsupervised type constraints on multi-hop complex logical query processing. Moreover, it can capture the different representations of type features when entities are in different locations in the logical path. We designed type compatibility measurement meta-operators combined with popular Existential Positive First-Order (EPFO) neural logical operators to achieve type-aware EPFO complex query embedding. We validated the effectiveness of our framework on popular large-scale knowledge graphs by using the same embedding dimensionality as complex logical embedding methods. The results showed an average relative improvement of 1.9–12.8% on Hit@3 and up to 42.1% on the certain logical pattern.

Keywords: knowledge graph; knowledge representation; type constraints; knowledge reasoning; representation learning; unsupervised learning

Citation: Chen, D.; Li, Q.; Gu, J. UnTiCk: Unsupervised Type-Aware Complex Logical Queries Reasoning over Knowledge Graphs. *Electronics* **2023**, *12*, 1445. https://doi.org/10.3390/electronics12061445

Academic Editor: Fabio Grandi

Received: 17 January 2023
Revised: 12 March 2023
Accepted: 16 March 2023
Published: 18 March 2023

1. Introduction

Knowledge Graphs (KGs) record much of the established knowledge of the real world in a universal form without designing database fields. Such a form of data is widely used in large-scale databases such as Freebase [1], YAGO [2], and DBpedia [3]. The feature of knowledge graphs not having predefined schemas allows them to gain flexibility with fewer constraints, which makes reasoning difficult. Multi-hop reasoning is a popular new question on the knowledge graph. For a complex logical query, a simple recursion of the traditional single-hop models (or knowledge graph completion models) [4–9] would be ineffective and would not handle the combination logic of it well. A promising approach is to embed the elements of the knowledge graph and complex logical queries into a low-dimensional space, looking for entities' embedding of the knowledge graph close to the representation of query embedding results in the space to get answers. In contrast to a rule-based or path-based multi-hop reasoning solution [10–13], which requires modeling predefined rules and intermediate nodes, the logical query embedding methods require less consumption, yet have better generalization [14].

However, existing complex logical query embedding models inevitably introduce structurally similar wrong answers as they mainly capture the structural information of entities. A common solution for knowledge graph completion models is introducing more entity information, such as the entity type [15–18]. There is a need to introduce type constraints for the complex logical query embedding models to enhance their reasoning power. Due to the lack of explicit type information on large knowledge graphs, such constraints should be unsupervised and generalizable. This leads to a challenging task that combines unsupervised type constraint information and complex logical query embedding methods.

Some studies have modeled unsupervised type constraints on knowledge graph completion models [17–20]. They explored some approaches to add unsupervised type constraints on single-hop completion models and mentioned the influence of the relations and location on entity types. However, we cannot directly generalize their approaches because complex logical query embedding models do not model intermediate nodes of the logical path.

Let us consider a logical question, "*What are the albums of the singers who won the Grammy Awards?*" There are two focus issues: First, complex logical query embedding methods relying only on entity structure information may return singles, movie promos, or even more answers with similar, but incompatible entity types. Second, there is an essential question that many entities, in reality, will exhibit different bias-type representations when faced with different relations and in different roles of the relation. A person who wins a Grammy as a singer can be an actor, a director, or even an athlete. The *diversity of entity types* can be even more difficult in multi-hop complex logical embedding.

Based on the above thoughts and issues, we propose UnTiCk. As its name implies, our framework can *untick* entities whose types do not conform to the constraints. In brief, we designed four *type compatibility measurement meta-operators*, which we simply refer to as *type meta-operators* in the following, by combining the four type meta-operators with neural logical operators containing only entity structure information. The implementation converts complex logical queries into type-aware complex logical queries. We used the type-aware neural logical operators recursively based on the computation graph to obtain the final set containing the type-aware answers. We successfully implemented unsupervised type constraints under the task of multi-hop complex logical query embedding. Furthermore, our approach captures different bias-type representations of entities for different relations and locations in the path.

Our contributions are as follows:

- We propose UnTiCk, an embedding-based unsupervised type-aware complex logical queries reasoning model. It is a novel solution that extends unsupervised type constraints to multi-hop complex logical query embedding models.
- We designed four type compatibility measurement meta-operations that reflect good modularity and generalization. They capture the diversity of entity types in different relations and locations in complex logical queries.
- We conducted experiments on three popular benchmark datasets, combining our model with popular complex logical embedding models. With the same number of embedding dimensions, our models showed better results than the complex logical embedding models, which contain only entity structure information. We also demonstrate the effectiveness of our unsupervised type feature extraction with a visualization.

2. Related Work

Our framework was inspired by two lines of prior work: logical query embedding models for multi-hop complex logical reasoning tasks and unsupervised type constraint methods in single-hop knowledge graph completion tasks. This section mainly introduces two lines of related work described above. At the beginning of each subsection, we present additional instructive background work in related fields. We summarize the most-relevant work in Table 1.

Table 1. Main related work and classification.

Fields	Model Name	Model Category		Type Constraint	
		Logical Query	Type Information	Unsupervised	Supervised
Logical Query Embedding	GQE [14]	Point	–	–	–
	Query2Box [21]	Box region	–	–	–
	NewLook [22]	Box region	–	–	–
	BetaE [23]	Beta distribution	–	–	–
Type Information Embedding	TypeDM and TypeComplex [19]	–	Entity-relation matching	✓	
	CooccurX [20]	–	Entity-relation matching	✓	
	ProtoE [17]	–	Entity-relation matching	✓	
	AutoETER [18]	–	Relation-specific extraction	✓	
	TEMP [24]	Plug-in module	Message passing		✓

2.1. Logical Query Embedding Models for Multi-Hop Reasoning

A popular and effective approach to solving incomplete knowledge graph problems is to embed knowledge graphs into a low-dimensional space. Early knowledge graph embedding models [4–7] brought better results on the knowledge graph completion task. Later models [8,9] extended the embedding to the complex space to model more relation patterns, but they can also handle only single-hop problems.

Some models [25–27] use the output of the embedding model as a guide, combined with posterior computation, to find complex queries' approximate answers. However, using the edge prediction model and scoring the possible paths, the required computational time is exponential in the number of existentially quantified variables in the query [28]. Moreover, since the model is not trained end-to-end, it cannot be optimized for error propagation between the model and the algorithm.

In recent years, researchers in the knowledge graph community have gradually shifted their attention from single-hop knowledge graph completion tasks to complex query reasoning tasks. They have made significant progress in the field of complex logical query embedding.

The logical query embedding methods [14,21–23,29,30] are promising approaches for solving multi-hop logical query reasoning tasks. Due to their performance, recent work used them for the KGQA task [31]. They avoid modeling intermediate entities while possessing greater flexibility and generalization capabilities than the traditional rule- and path-based multi-hop reasoning methods [10–13]. We will introduce representative work in this field.

GQE [14] models the whole logical query process in a low-dimensional space, treats entities as points, and represents logical operators as learned geometric operations in the embedding space, implementing complex query reasoning on the subset of First-Order Logic (FOL).

Query2Box [21] is currently a popular and intuitive model in the field of logical query embedding, with the best balance of computational efficiency, model complexity, and accuracy. Its main idea treats anchor entities as points (box regions with zero offset) in a low-dimensional space, but maps logical operators and predicted results to box regions, similar to Venn diagrams [32]. Consistent with our intuition about multi-hop logical query reasoning, we applied a smaller weight to the distance inside the box so that it is dominant in the ranking, meaning it is closer to the answer.

NewLook [22] reduces the cascading error on the Query2Box projection operation by introducing nonlinearity while supporting difference operations and multivariate node queries. However, using randomly sampled adjacency matrices has some impact on spatial complexity.

BetaE [23] models uncertainty in complex logic queries with a beta distribution while supporting a complete set of first-order logical operators. The cost is that it increases the complexity of the model and is not intuitive enough compared to box embedding.

2.2. Unsupervised Type Information Embedding Models

Many single-hop models that utilize entity types require entity annotations or explicit type prior knowledge.

For the knowledge graph completion task, TKRL [15] models explicit hierarchical type information, and TransT [33] integrates type information through probabilistic estimation. They both propose that entities should have multiple representations in different types. On the entity type inference task, ConnectE [16] models not only entity types, but also type triples. CORE [34] embeds entities and types into the complex space separately and connects the two spaces by a regression model. TET [35] designs three transformers to capture information, which provides a new way for the entity type inference task with Transformers [36]. However, the above work cannot satisfy our need for complex logical queries.

TEMP [24] extends the application of entity type information to multi-hop reasoning. It is implemented through a combination with existing query embedding models, using message passing to aggregate explicit type information. The information requires much manual effort and is not always available on modern large knowledge graphs. This is different from the goal of our model.

We wanted to model unsupervised type constraints of knowledge graphs to make them applicable to a wider range of situations. Some prior work has explored this in the single-hop tasks, which is the focus of this subsection.

They are mainly based on two ideas, which we classify as entity-relation matching and relation-specific extraction. The former assigns type information vectors for each entity and relation separately and achieves type constraints by matching scores. The latter designs extraction matrices for each relation and applies a translation mechanism to the extracted type representations to score them to achieve type constraints. The specific categories of the models are shown in Table 1.

TypeDM and TypeComplex [19] proposes a type-sensitive method based on the popular knowledge graph completion models DistMult and ComplEx. It does not require prior knowledge and measures type compatibility by adding a new vector space. TypeDM and TypeComplex designs four embedding vectors for each tuple (s, r, o), representing the type of subject entity, type of object entity, expected type of subject entity, and object entity by the relation. It performs type-sensitive constraints by computing a joint score function of subject type compatibility, object type compatibility, and the original knowledge graph model score.

CooccurX [20] proposes co-occurrence to calculate the type similarity between entities. The main idea is that a pair of entities with more simultaneous subjects or objects in the same relation is more similar in type. However, this computation cannot be applied to large-scale knowledge graphs.

ProtoE [17] is similar to TypeDM and TypeComplex, except that it embeds type embedding and entity embedding in the same low-dimensional space. It considers the diversity of prototypes of head and tail entity types and redesigns the loss function. This approach extends unsupervised type constraints to more knowledge graph completion models. It brings more parameters while still only handling single-hop completion problems.

AutoETER [18] proposes a joint entity and unsupervised type embedding model. It sets up the entity embedding as a complex space embedding similar to RotatE, where the embedded entities and relations are in the complex space. It treats the relations as rotations between the head and tail entities. When extracting unsupervised type constraints, the type representations of head and tail entities are extracted in space using relation-specific feature extraction matrices while treating the relation vectors as type transformations. This process is most relevant to our idea. Unlike the diversity of head and tail entity type prototypes in ProtoE, we should intuitively pay more attention to the diversity of type representations of entities in the face of different relations, because entities usually show specific features for particular relations. However, we focused on type compatibility measurement in multi-hop logical query embedding. In contrast, they focused on multi-type representations of entities in knowledge graph completion tasks without any logic and path.

3. Background and Problem Definition

In this section, we first define the concepts and operators used in this paper, then briefly introduce the popular complex logical embedding models used, finally leading to a formal statement of the problem we expected to solve. We need a clear definition of the knowledge graph at the beginning of this section.

Definition 1 (Knowledge graph)**.** *A knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ consists of entities $e \in \mathcal{E}$, relations $r \in \mathcal{R}$, and tuples (or triples) $(h, r, t) \in \mathcal{T}$. h represents the head entity of the triple, and t represents the tail entity of the triple connected by relation r. We consider $r(h, t) = \{1, 0\}$ to be a binary function representing $(h, r, t) \in \mathcal{T}$ when $r(h, t) = 1$ and $(h, r, t) \notin \mathcal{T}$ otherwise.*

The point of this work is unsupervised type constraints in complex logical query embedding. It is necessary to formalize the details of the unsupervised type constraints we need to capture, even if they are not explicitly identified in datasets.

Definition 2 (Type constraint collection of knowledge graph)**.** *A type constraint collection $\mathcal{G}^{\mathcal{T}}$ of a knowledge graph $\mathcal{G}$ is denoted as $\mathcal{G}^{\mathcal{T}} = (\mathcal{E}^{\mathcal{T}}, \mathcal{R}^{\mathcal{T}}, \mathcal{C}^{\mathcal{E}}, \mathcal{C}^{\mathcal{R}})$, where $\mathcal{E}^{\mathcal{T}}$ is the constraint distinct type set of $\mathcal{E}$ and $\mathcal{R}^{\mathcal{T}}$ is the distinct type set of $\mathcal{R}$, $\mathcal{C}^{\mathcal{E}}$ is a constraint function that $\mathcal{C}^{\mathcal{E}}(e) = \{c_1, c_2, ..., c_e\}$ represents the distinct type set of entity e. $\mathcal{C}^{\mathcal{R}}$ is a constraint function that $\mathcal{C}^{\mathcal{R}}(r) = \{(c_{h_1}, c_{t_1}), (c_{h_2}, c_{t_2}), ..., (c_{h_n}, c_{t_n})\}$ represents the head (domain) and tail (range) distinct type constraints tuple of relation r.*

Following the definition of previous works [21], the complex logical query domain of this article is at Existential Positive First-Order (EPFO) logical queries (i.e., queries that include any set of $\wedge$, $\vee$, and $\exists$).

Some recent articles attempted to solve logical query embedding on a complete set of first-order logical operators. BetaE [23] can model negation using probabilistic inversion, as it models logical operations natively as beta distributions. NewLook [22] can solve negation using the whole set because it models the difference logical operator.

However, these models are similar to model arbitrary quantifiers in that they cause a dramatic increase in the query space. They are less used in practice, and we often solve them by combining and excluding multiple queries, which sometimes leads to faster ones. This paper considered a common generic method in EPFO queries. These modeling differences are not the focus of this paper, so they will not be expanded further here.

Definition 3 (EPFO logical queries)**.** *We formally define an EPFO logical query as follows:*

$$\begin{aligned} & q[a_?] = a_? \,.\, \exists e_1, ..., e_k : p_1 \wedge p_2 \wedge ... \wedge p_n, \\ & \text{where } p_i = r(e_a, e) \text{ or } r(e', e), e_a \in \mathcal{E}_a, r \in R, \\ & \qquad e \in \{a_?, e_1, ..., e_k\}, e' \in \{e_1, ..., e_k\}, e \neq e', \end{aligned} \tag{1}$$

where $a_?$ is our target variable, e_a represents the anchor entity node of the query, which is a non-variable node, $e_i, ..., e_k$ are existentially quantified bound variables, and p_i represents the atomic formula. An atomic formula represents the projection of a relation between an anchor entity node and an existentially quantified bound variable or two existentially quantified bound variables.

Given an EPFO logical query, we want to find the answer set $[\![q]\!] \subseteq \mathcal{E}$ s.t. $\forall e \in [\![q]\!] \Leftrightarrow q[e] = 1$.

A common approach is to convert logical queries into corresponding dependency graphs and computation graphs to guide the embedding model for the result query process [14,21–23,37]. We define them as follows.

Definition 4 (Dependency graph and computation graph)**.** *A complex logical query q has a corresponding dependency graph denoted as* $\mathcal{Q} = (\mathcal{E}_{\mathcal{Q}}, \mathcal{R}_{\mathcal{Q}}, \mathcal{L}_{\mathcal{Q}})$*.* $\mathcal{Q}$ *consists of entities* $\mathcal{E}_{\mathcal{Q}} \subseteq \mathcal{E}$*, which contain the anchor nodes' set and the existentially quantified bound variables' set. The relation between entities* $\mathcal{R}_{\mathcal{Q}} \subseteq \mathcal{R}$ *and* $\mathcal{L}$ *is the logical operator. In this paper, we considered the logical collection* $\mathcal{L}_{\mathcal{Q}} \subseteq \mathcal{L} = \{\mathbb{P}, \mathbb{I}, \mathbb{U}\}$ *(project, intersection, union operators). The computation graph represents the actual order of operations of our framework for reasoning about the dependency graph.*

A dependency graph for a valid query must be a Directed Acyclic Graph (DAG) with the source node as the anchor node and the target node as the unique sink node [37]. Let us consider a typical question that is more convenient to give an example: "*Who are the singers of the songs composed by Jay Chou and lyrics written by Vincent Fang?*" The corresponding dependency graph and computation graph for the complex logical query is shown in Figure 1.

$$q = A_? \,.\, \exists V : isComposerOf(\text{Jay Chou}, V) \wedge isLyricistOf(\text{Vincent Fang}, V) \wedge isSangBy(V, A_?)$$

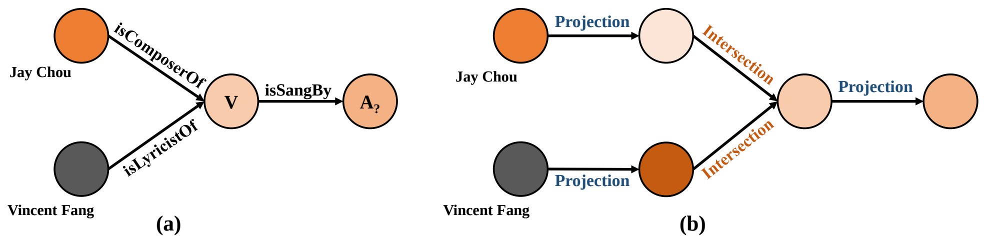

Figure 1. This is the graph of the complex logical query "*Who are the singers of the songs composed by Jay Chou and lyrics written by Vincent Fang?*" (**a**) Dependency graph for the above query. (**b**) Computation graph for the above query.

In summary, we now conceive of the model to focus on extracting the corresponding collection of type constraints during the traditional embedding of complex logical queries, as well as how to convert EPFO queries into type-aware EPFO queries and execute joint queries based on dependency graph and computation graph. Based on this idea, we wanted to design some assemblable type meta-operator components. We combined them with the neural logical operators of the logical query embedding to achieve this goal.

In the following, we first introduce the similarities and differences of several popular EPFO complex logical query embedding models from the perspective of logical embedding operators. They will be used as part of our model to capture the original entity structure information of the knowledge graph. Then, we define the problems of the type-aware neural logical operators. Finally, we obtain a formal representation of the complete problem that UnTiCk needs to solve.

3.1. Logical Query Embedding Operators

It is necessary for us in this subsection to briefly describe the intuition and design of GQE [14] and Query2Box [21] for the entity logical embedding operator, which will be effectively integrated into our framework as the basis for entity structure information embedding. We summarize their main features and the way the logical query embedding operator is implemented and briefly introduce their main ideas in the following.

Query2Box implements all the logical operators required by EPFO queries, which can be seen as an extension of the idea of GQE, and we decided to introduce its design first. It uses zero offset box embedding to represent an entity (which can also be regarded as a point), box transformations to represent relations, and the result box region to represent the answer prediction space for a logical query. The box embedding is defined as:

$$\mathbf{b} = \{\mathbf{e} \in \mathbb{R}^{d_e} : \mathbf{b^c} - \mathbf{b^o} \leq \mathbf{e} \leq \mathbf{b^c} + \mathbf{b^o}\}, \qquad (2)$$

where $\mathbf{e}$ is the embedding of entity $e \in \mathcal{E}$, d_e is the embedding dimensionality of e, and $\mathbf{b}$ is the box region consisting of center $\mathbf{b^c}$ and offset $\mathbf{b^o} \geq \mathbf{0}$.

When initializing an anchor node, since it contains only one entity, intuitively, Query2Box represents it as a box with zero offset. The model then uses logical query embedding operators to obtain the final answer region based on the computational graph. The target entity is located at or close to the final box region.

Entity projection operator: Given an entity embedding or an existential quantified variable box region and a relation box embedding $\mathbf{r}$, we model the entity projection operation as a linear transformation:

$$\hat{\mathbf{b^c}} = \mathbf{b^c} + \mathbf{r^c}, \hat{\mathbf{b^o}} = \mathbf{b^o} + \mathbf{r^o}, \tag{3}$$

where $\mathbf{r^c}$ and $\mathbf{r^o}$ are the center and offset of the relation box embedding $\mathbf{r}$ and $\hat{\mathbf{b^c}}$ and $\hat{\mathbf{b^o}}$ form the box region $\hat{\mathbf{b}}$, which is the prediction of the box embedding space after the projection operation.

Entity intersection operator: Given a collection of entity sets, the entity intersection operation's intuition is to find the overlap of multiple spaces. Since the importance of each collection element is not the same for the merged space, it uses the attention mechanism to find the entity centers and the *sigmoid* function of *Deepsets* [38] to shrink the offset:

$$\hat{\mathbf{b^c}} = \sum_i \mathbf{a_i} \odot \mathbf{b_i^c}, \mathbf{a_i} = \frac{exp(MLP(\mathbf{b_c^i}))}{\sum_j exp(MLP(\mathbf{b_j^c}))}, \tag{4}$$

$$\hat{\mathbf{b^o}} = Min(\{\mathbf{b_1^o}, ..., \mathbf{b_n^o}\} \odot \sigma(Deepsets(\{\mathbf{b_1^o}, ..., \mathbf{b_n^o}\})), \tag{5}$$

$$Deepsets(\{\mathbf{b_1^o}, ..., \mathbf{b_n^o}\}) = MLP(\frac{1}{N}\sum_{i=1}^{N} MLP(\mathbf{b_i^o})), \tag{6}$$

where $\odot$ is the dimensionwise product, $MLP(\cdot)$ is the multi-layer perceptron, and σ represents the *sigmoid* function; $Deepsets(\cdot)$ follows the idea of GQE to guarantee permutation invariance, shown in Equation (6).

Entity union operator: Given a collection of entity sets, the entity union operation's intuition is to transform the query into a Disjunctive Normal Form (DNF) [39], instead of creating a new neural logical operator. In the final step, the box regions of all anchor nodes need to be united, and the shortest distance in the answer space will determine the confidence of the entity:

$$S_{agg}(\mathbf{e}, q) = Min(\{S_1(\mathbf{e}, \mathbf{q_1}), ..., S_1(\mathbf{e}, \mathbf{q_n})\}), \tag{7}$$

where q is the complex logic query, $\mathbf{q_i}$ is every single path of the union query, $S_{agg}(\cdot)$ represents the confidence function of the entity answer after the union operation, and $S_1(\cdot)$ represents the score function to calculate the distance of the entity representation from the center of the predicted box region.

It solves the problem that the answer space is not closed and makes the model only need to be united in the last step. Moreover, if not converted to the DNF, the model may need to model the *powerset* of the entity set in vector space, which is unacceptable in many cases, much less to support all union queries.

Finally, they use the following distance function to calculate the confidence level of the entity as an answer:

$$S_1(\mathbf{e}, \mathbf{q}) = dist_{outside}(\mathbf{e}, \mathbf{q}) + \alpha \cdot dist_{inside}(\mathbf{e}, \mathbf{q}), \tag{8}$$

where $dist_{inside}(\cdot)$ and $dist_{outside}(\cdot)$ are the inside and outside parts of L_1 *distance* between the entity embedding and box center, respectively. $\alpha \in (0, 1)$ is a hyperparameter that makes elements inside the box have a smaller distance and gives a larger penalty to entities outside the box. It gives the entities inside the box a greater propensity to be the answers.

We used the extended implementation of GQE [21], making it work in the EPFO query space. It models entities as points, and the entity projection operator is modeled as translation. The entity intersection operator is modeled by *Deepsets*. It also uses the DNF-query rewriting strategy and L_1 *distance* as a score function to calculate the score between entity embedding $\mathbf{e}$ and query box $\mathbf{q}$.

3.2. Type-Aware Logical Query Operations

According to the above, our problem is to convert existing logical query embedding operators to type-aware neural logical operators. We define three type-aware neural logical operations as follows.

Problem 1 (Type-aware projection)**.** *Given a set of entities $\mathcal{E}_q \subseteq \mathcal{E}$, relation $r \in \mathcal{R}$, and their corresponding type constraint collection $\mathcal{G}_q^{\mathcal{T}}$, for $e_h \in \mathcal{E}_q$, we use a type-aware projection operation to obtain the answer collection $\mathbb{P}_t(e_h, r) = \{e_t | r(e_h, e_t) = 1, \exists (c_h, c_t) \in \mathcal{C}^{\mathcal{R}}(r), c_h \in \mathcal{C}^{\mathcal{E}}(e_h), c_t \in \mathcal{C}^{\mathcal{E}}(e_t)\}$.*

Problem 2 (Type-aware intersection)**.** *Given a collection of entity sets $E = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n\}$, the corresponding entity type constraint set $C = \{\mathcal{C}_1^{\mathcal{E}}, \mathcal{C}_2^{\mathcal{E}}, ..., \mathcal{C}_n^{\mathcal{E}}\}$. The prediction constraint set after the intersection is $\mathcal{C}_{\mathbb{I}}^{\mathcal{E}}$. Then, we use the type-aware intersection operator to obtain the answer collection $\mathbb{I}_t(E, C) = \cap_{i=1}^{n} \{e \in \mathcal{E}_i | \exists c_e \in \mathcal{C}_i^{\mathcal{E}}(e), c_e \in \mathcal{C}_{\mathbb{I}}^{\mathcal{E}}\}$.*

Problem 3 (Type-aware union)**.** *Given a collection of entity sets $E = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n\}$, the corresponding entity type constraint set $C = \{\mathcal{C}_1^{\mathcal{E}}, \mathcal{C}_2^{\mathcal{E}}, ..., \mathcal{C}_n^{\mathcal{E}}\}$. The prediction constraint set after the union is $\mathcal{C}_{\mathbb{U}}^{\mathcal{E}}$. Then, we use the type-aware union operator to obtain the answer collection $\mathbb{U}_t(E, C) = \cup_{i=1}^{n} \{e \in \mathcal{E}_i | \forall c_e \in \mathcal{C}_i^{\mathcal{E}}(e), c_e \in \mathcal{C}_{\mathbb{U}}^{\mathcal{E}}\}$.*

To solve the problems, we carefully analyzed the common operations required by the logical query embedding model to implement type-aware operators. The logical query embedding model can be combined by designing type compatibility measurement meta-operators to perform joint metrics with it for type information.

3.3. UnTiCk Problem Definition

Based on the above definitions, we can formalize the unsupervised type constraints logical query embedding problem described in this paper as follows.

Given a knowledge graph $\mathcal{G}$, an EPFO logical query q, and type compatibility measurement meta-operators set, we need to extract the type constraint information from it. Then, we use the type-aware neural logical operators recursively through the computation graph generated by the dependency graph. Finally, we obtain a set of answer entities $[\![q^t]\!]$ that conform to the type constraint information and logical structure information.

4. UnTiCk: Unsupervised Type-Aware Complex Logical Queries Reasoning Framework

We designed a joint framework to combine the popular complex logical embedding framework for the entity structure information embedding module. We also designed the entity type information embedding module. It consists of type compatibility measurement meta-operators, which we will describe in detail below. An overview of our UnTiCk framework is shown in Figure 2.

In the model's training process, we jointly trained the entity structure information embedding module and the entity type information embedding module and implemented type-aware complex logical query embedding after the joint evaluation of the two modules. We briefly introduce the query processing process of our framework with typical complex logical queries in the following.

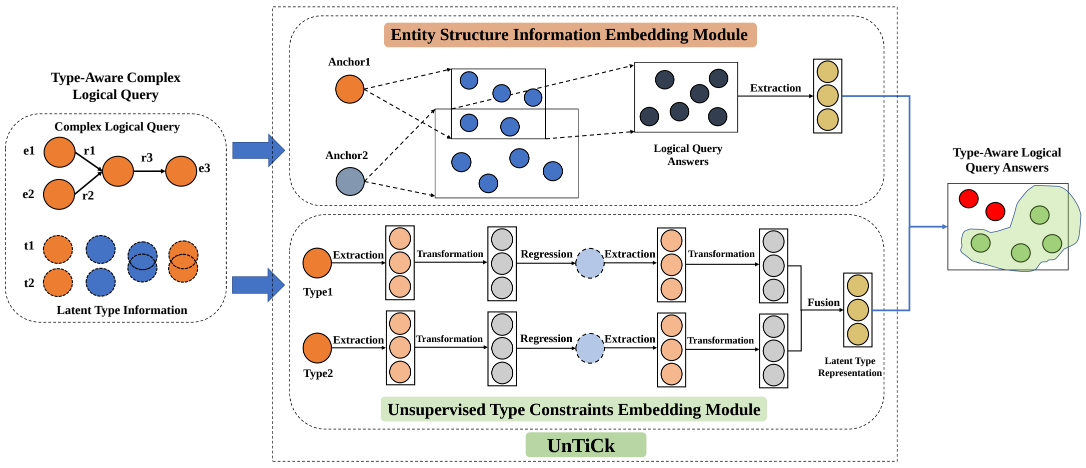

Figure 2. The overview of the UnTiCk framework.

4.1. Type Compatibility Measurement Meta-Operators

Based on the definitions in the previous section, we analyzed the intermediate states and measurement methods for unsupervised entity types during logical query reasoning. Then, we summarize them into four different type compatibility measurement meta-operators, which we design concretely in the following.

We represent the unsupervised types of entities as separate points because unsupervised type constraints do not have an explicit type separation. Moreover, our goal was not to perform a classification task, but to constrain our target entities in an additional dimension. We only need to focus on the different features of an entity in the face of a particular relation and the specific degree of expression that captures these features.

It is a smaller vector space compared to the entity structure embedding space, allowing entity and relation representations to show more group similarity features in lower dimensionality.

Let us continue to consider the typical question: *"Who are the singers of the songs composed by Jay Chou and lyrics written by Vincent Fang?"* We illustrate our type compatibility measurement meta-operator design with an example type knowledge graph. It contains the previous question. For each meta-operator, we used the part of the knowledge graph that best represents its characteristics as an example to describe it. The specific structure of the type knowledge graph is shown in Figure 3. We describe our approach with a phased example of the above problem.

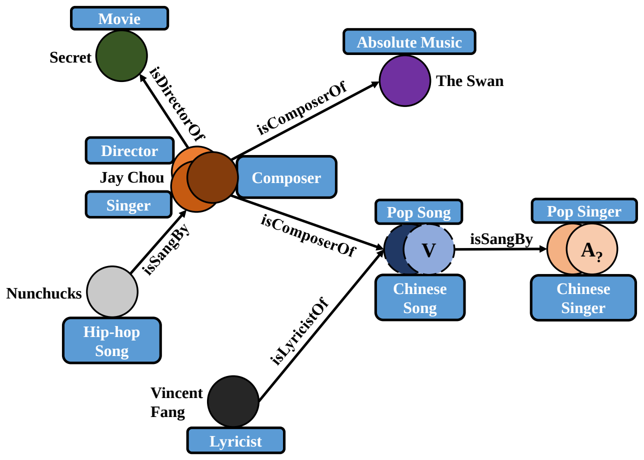

Figure 3. This is an example of the type knowledge graph used in this section. It shows the entities and relations and the different types of entities. V represents the virtual intermediate node. $A_?$ represents the target node.

Example 1. *For the anchor node **Jay Chou** , when it is in **(Jay Chou, isComposerOf, The Swan)**, its entity type feature is expressed as **Composer**. In **(Jay Chou, isDirectorOf, Secret)**, its entity type feature is expressed as **Director**. In **(Nunchuncks, isSangBy, Jay Chou)**, its entity type feature is expressed as **Singer**.*

With Example 1, we need to design a meta-operator that makes an entity exhibit different bias type representations in the face of different relations. We refer to this meta-operator as *type feature extraction* and define it as follows.

Definition 5 (Type feature extraction)**.** *Given an entity e and an outgoing relation r, $Ex(e,r)$ aims to obtain the domain type representation of this entity $Ex(e,r) = \{c_h | \exists(c_h, c_t) \in \mathcal{C}^{\mathcal{R}}(r)$, $c_h \in \mathcal{C}^{\mathcal{E}}(e)\}$ for this outgoing edge r.*

For the design of the type feature extraction function, our intuition is to design a relation-specific representation matrix for each relation:

$$\mathbf{c_e} = \mathbf{W_r t_e} \tag{9}$$

where $\mathbf{c_e}$ represents the relation-specific type representation of the entity e, $\mathbf{t_e}$ represents the type representation of the entity e, and $\mathbf{W_r}$ represents the relation-specific representation matrix for relation r. For an entity as the outgoing edge of a relation, $\mathbf{c_e}$ represents the domain type representation for relation r. Similarly, for an entity as the incoming edge of a relation, $\mathbf{c_e}$ represents the range type representation for relation r.

The representation matrix represents the unique preference of each relation for the entity type. It leads each entity to exhibit a different type representation for different relations. Meanwhile, the entity's representation needs to be converted to accommodate complex logical paths, as it serves as both the head of one relation and the tail of another. It also brings new challenges for the other meta-operators' design.

Example 2. *For the anchor nodes **Jay Chou** and **Vincent Fang** in the previous example, we should not show modeling specific intermediate nodes during reasoning, but we need to obtain intermediate nodes as range type constraint representations of **isComposerOf** and **isLyricistOf**.*

With Example 2, we find that we need to design a meta-operator that converts the entity relation-specific domain representation we have obtained into an entity relation-specific range representation. We call this meta-operator *type feature transformation* and define it as follows.

Definition 6 (Type feature transformation)**.** *Given a domain entity type representation c_h^e and an outgoing relation r, $Tr(r)$ aims to obtain the range type representation of this entity e for this outgoing edge r, denoted as $Tr(c_h^e, r) = \{c_t | (c_h^e, c_t) \in \mathcal{C}^{\mathcal{R}}(r)\}$.*

The representation of a relation in the type embedding space can be regarded as a transformation from its domain type to its range type. The type feature transformation function is represented as a linear transformation:

$$\mathbf{c_t} = \mathbf{c_e} + \mathbf{t_r}, \tag{10}$$

where $\mathbf{c_e}$ represents the domain embedding of entity e extracted by relation r and $\mathbf{t_r}$ represents the type transformation of r from its domain to the range type. We used this to obtain the range feature $\mathbf{c_t}$ after transformation.

If we obtain the predicted range type representation of the target node at this point, we can directly perform type feature extraction on the target entity node and calculate the score. Alternatively, if we obtain the predicted range type representation of the intermediate node at this point, we need a meta-operator to connect the intermediate nodes before and after.

This meta-operation can be regarded as the inverse operation of type feature extraction. However, it is obvious that taking the inverse matrix of learnable parameters for type feature extraction is not feasible in all cases. Furthermore, we cannot handle the cascading errors introduced by simple inverse operations in long paths.

Here are two possible approaches. The first one is to use a feedforward neural network to directly fit the process of inputting the range of one relation and outputting the domain of another relation. The second one is to use the feedforward neural network to fit a virtual node that serves as a placeholder intermediate node on this type-constrained path to allow reasoning to proceed.

We chose the latter one because the former approach has two limitations: First, a simple feedforward neural network may not be expressive enough in this case. The range representation we obtain will already have some loss of original entity type features, and it may introduce large errors due to the arbitrary combination of relations connected before and after the intermediate nodes. Second, this approach loses the atomicity of the meta-operators. It actually assumes fitting both regression and extraction meta-operations, which violates our original design principle.

We still use an example to demonstrate this process.

Example 3. *Once we have obtained the range representation of* ***isComposerOf*** *and* ***isLyricistOf*** *in the query, we need to regress the virtual nodes' representation from the range representation for each logical path. These nodes refer to the original features of the entity node composed by* ***Jay Chou*** *and written lyrics by* ***Vincent Fang****, and they will be used as the following Relation* ***isSangBy*** *for type feature extraction.*

With Example 3, we need to design a meta-operator that converts the entity relation-specific range representation into a virtual node representation. We call this meta-operator *type feature regression*, defined as follows.

Definition 7 (Type feature regression)**.** *Given a relation r and a range entity type representation c^v of the intermediate virtual node V for r, $Re(c^v, r)$ aims to obtain the original entity type representation of the intermediate virtual node V, $Re(c^v, r) \subseteq \{v | \exists (c_h, c^v) \in \mathcal{C}^{\mathcal{R}}(r), c^v \in \mathcal{C}^{\mathcal{E}}(v)\}$.*

At the intermediate node of a logical query, an entity will serve as both the tail of the preceding relation and the head of the following one. We already have type feature extraction and type transformation meta-operations. Now, the question is: How do we regress a relation-specific entity type representation to the original entity type representation without modeling the actual representation of the intermediate nodes? We designed the following type feature regression meta-operator:

$$\mathbf{v} = MLP(\mathbf{t_h} || \mathbf{t_r} || \mathbf{c_t}), \tag{11}$$

where $\mathbf{v}$ represents the original entity type representation of the intermediate virtual node V, $\mathbf{t_h}$ represents the original entity type representation of the previous node of the virtual intermediate node v (i.e., the head entity of the relation r), $\mathbf{c_t}$ represents the range embedding extracted by relation r, $\mathbf{t_r}$ represents the type transformation of r from its domain to the range type, and $||$ is a concatenation operation. We think we can handle this matter with a generic multi-layer perceptron.

The intuition is that the primitive representation of an intermediate node's entity type is strongly related to its domain entity representation, the relation transformation representation, and the relation-specific type representation of them. We concatenated these three representations together and output the final intermediate node primitive type representation using MLP.

In the type constraint reasoning process above, we maintained a separate predicted type constraint for each path. This is because, in terms of factual logic, the constraints for each different path should be independent. We computed the logical path to the target node by the above three meta-operations.

Example 4. *After the recursive computation in the previous example, we obtain the target node type representation of the logical paths where each of the two anchor nodes, **Jay Chou** and **Vincent Fang**, is located. We need to calculate the scores of each path and jointly find the target answers.*

With Example 4, the virtual target node type representations on the two paths jointly determine the different type constraints on the gold target nodes. They are intuitively part of this entity type. In other words, the virtual node does not represent a specific entity that exists in reality, but is only a placeholder with a logical feature reference.

In our EPFO logical query, we need to consider two operations requiring the fusion of multiple paths, the intersection and the union, in the last step. We need a meta-operator to obtain the final type constraint score. We call this meta-operator *type feature fusion* and define it as follows.

Definition 8 (Type feature fusion)**.** *Given the target node e, the list of incoming edges of the target node R, and the list of type constraints C for each logical path, $Fu(e, R, C)$ aims to obtain a composite score of the type constraints' compatibility for multiple logical paths after feature intersection or union (R and C identified as intersection or union).*

Continuing with the above typical example, suppose we have obtained the prediction type range representation of the two path sink nodes.

For each path, we used a feature extraction of the target node, then calculated the score with it and our predicted range type representation. For the intersection, each path is individually constrained, and we perform the mean value directly:

$$S_{int}(\mathbf{t_e}, \mathbf{W_{int}}, \mathbf{c_{int}}) = Mean(\{S_2(\mathbf{t_e}, \mathbf{W_1}, \mathbf{c_1}), ..., S_2(\mathbf{t_e}, \mathbf{W_n}, \mathbf{c_n})\}), \tag{12}$$

where $\mathbf{t_e}$ represents the original entity type presentation of the target node, $\mathbf{W_{int}}$ represents the intersection path list of relation-specific representation matrix for each logical path, $\mathbf{c_{int}}$ represents the intersection path list of predicted type constraints for each logical path, $Mean(\cdot)$ represents the arithmetic mean function, and $S_2(\cdot)$ represents the score function for the type constraint.

For the path of union, we moved it to the last step by converting the query to the DNF in the computation graph. Similar to the intersection, except we obtain the maximum value of the score for different paths:

$$S_{uni}(\mathbf{t_e}, \mathbf{W_{uni}}, \mathbf{c_{uni}}) = Max(\{S_2(\mathbf{t_e}, \mathbf{W_1}, \mathbf{c_1}), ..., S_2(\mathbf{t_e}, \mathbf{W_n}, \mathbf{c_n})\}), \tag{13}$$

where $\mathbf{W_{uni}}$ represents the union path list of the relation-specific representation matrix for each logical path, $\mathbf{c_{uni}}$ represents the union path list of predicted type constraints for each logical path, and $Max(\cdot)$ represents the maximum value function.

When there are multiple anchor nodes in the computation graph, we first treat its type constraints as multiple independent type constraint paths. In the EPFO query, we need to consider the joint operation between two paths, intersection and union. We recorded the order in which the intersection and union operations appear during the path reasoning process and perform a type feature fusion in the final stage.

We used the L_1 *distance* between the relation-specific type representation of the target node and the predicted type constraint as the scoring function for type compatibility:

$$S_2(\mathbf{t_e}, \mathbf{W}, \mathbf{c}) = ||\mathbf{W}\mathbf{t_e} - \mathbf{c}||_1. \tag{14}$$

where $|| \cdot ||_1$ represents the L_1 *distance* function. We perform type feature extraction on the candidate nodes and score the ranking with our calculated type representation of the target nodes. Nodes with smaller distances are more consistent with the unsupervised type constraints of the logical query.

We modeled type constraint in the complex logical query process by designing the above four type meta-operators. Next, we describe how to apply the type meta-operators and the logical query embedding operators to implement type-aware neural logical operators.

4.2. UnTiCk Query Reasoning Process

Based on the previously described logical query embedding operators and type compatibility measurement meta-operators, we iteratively used type-aware neural logical operators, which consist of logical query embedding operators and type compatibility measurement meta-operators. We used the two-anchor-intersection-projection query structure of Figure 1 as an example to introduce our query embedding generation process. The details of this process are given as Algorithm 1. We use $\{\mathbb{P}, \mathbb{I}, \mathbb{U}\}$ to represent the structure projection, intersection, and union operators, and $\{\mathbb{P}_t, \mathbb{I}_t, \mathbb{U}_t\}$ represents type-aware projection, intersection, and union operators below.

Algorithm 1: UnTiCk query embedding generation

```
Input: Query anchor nodes $\mathcal{E}_a$, query variable nodes $\mathcal{B}$, query edges $\mathcal{E}_q$, a map $d_q$
       from query variables to their degree in the query DAG
Output: Type-aware query embedding $\mathbf{q^t}$
$Q_s$, $Q_t$ = dictionary mapping every $\mathcal{V}_i \in \mathcal{B}$ to an empty set for structure and type;
for $r(e_i, V_j) \in \mathcal{E}_q : e_i \in \mathcal{A}$ do
    // One time of the loop as a type-aware projection.;
    $Q_s[V_j] = Q_s[V_j] \cup \mathbb{P}(e, r)$;
    $Q_t[V_j] = Q_t[V_j] \cup Tr(Ex(e, r), r)$;
end
$A_t$ = empty dictionary;
while $|Q_s.key\_set| > 0$ do
    $A_s$ = empty dictionary;
    // A loop as a type-aware intersection without fusion.;
    for $V_i \in Q_s.key\_set : |Q_s[V_i]| = d_q(V_i)$ do
        $A_s[V_i] = A_s[V_i] \cup Q_s[V_i])$;
        $A_t[V_i] = A_t[V_i] \cup Ex(Re(Q_t[V_i])), r)$ ;
        // relation $r$ in last projection.;
        delete $Q_s[V_i]$;
    end
    $A_s[V_i] = \mathbb{I}(Q_s[V_i]))$;
    for $V_i \in A.key\_set$ do
        for $r(V_j, V_k) \in \mathcal{E}_q : V_j = V_i$ do
            $Q_s[V_k] = Q_s[V_k] \cup \mathbb{P}(A_t[V_i], r)$;
            $Q_t[V_k] = Q_t[V_k] \cup \mathbb{P}(A_t[V_i], r)$;
        end
    end
end
// This step for type fusion.;
$Q_t[V_k] = Fu(Q_t[V_k])$;
return $A_s[a_?] \cap A_t[Ex(a_?, r)]$;
```

Next, we introduce the process of the combination of each type-aware operator and give proofs.

Type-aware projection operation: In the entity structure information embedding module, we perform one entity projection operation on it. In the entity type information embedding module, a relation feature extraction operation and a type feature transformation operation are performed on it if our target entity is an anchor node. It will add a type feature regression meta-operation before this if our target entity is an intermediate node.

Lemma 1. *In the logical patterns discussed in the experiments of this paper, the above solution is equivalent to Problem 1. (We prove this with a simple example: given a two-hop path query, the anchor entity is called e_1, the target entity is called e_3, and the relation is r_1, r_2, respectively.)*

Proof. According to the structural projection operation, we obtain the projection set:

$$\mathbb{P}(e_1, r) \Leftrightarrow \{e | r(e_1, e) = 1\}. \tag{15}$$

We can obtain a virtual intermediate node representation after applying Definitions 5–7:

$$v \Leftrightarrow \mathbb{P}(e_1, r_1) \cap Re(Tr(Ex(e_1, r_1), r_1), r_1), \tag{16}$$

then we apply Definition 5 to v and e_2, and we simply obtain:

$$\mathbb{P}_t(\mathbb{P}_t(e_1, r_1), r_2) \Leftrightarrow \mathbb{P}(v, r_2) \cap Tr(Ex(v, r_2), r_2); \tag{17}$$

this proves that our algorithm on this logical model is equivalent to using the type-aware embedding operator twice. □

Type-aware intersection operation: In the entity structure information embedding module, we perform one entity intersection operation on it. In the entity type information embedding module, each different incoming relation is a different constraint for the sink node of the intersection operation, and we should not ignore this information. Therefore, we record different path type compatibility separately, add the paths that need to be intersected with the intersection path list, and record the relevant information in the computation order list. We handle all this in the final path fusion operation.

Lemma 2. *In the logical patterns discussed in the experiments of this paper, the above solution is equivalent to Problem 2. (We prove this with a simple example: given a three-intersection query, the anchor entity set E contains e_1, e_2, e_3, the target entity is called e_4, and the relation is a list R with r_1, r_2, r_3.)*

Proof. According to the structural projection operation, we considered a one-hop path query. Applying Definition 5 to anchor entity e and target entity e' with relation r, we obtain:

$$\mathbb{P}_t(e, r) \Leftrightarrow \mathbb{P}(e, r) \cap Tr(Ex(e, r), r); \tag{18}$$

we also have:

$$\mathbb{I}(E) \Leftrightarrow \cap_{i=1}^{3} \mathbb{P}(e_i, r_i), \tag{19}$$

then using the proven type-aware projection operator for all three anchor nodes, we obtain type constraint list C, and finally, we can obtain:

$$Fu(e_4, R, \mathbb{I}_t(E, C)) \Leftrightarrow Fu(e_4, R, \mathbb{I}(E) \cap (\mathbb{P}_t(e_1, r_1) \cup \mathbb{P}_t(e_2, r_2) \cup \mathbb{P}_t(e_3, r_3))); \tag{20}$$

this demonstrates that the scores calculated based on our algorithm are consistent with using three type-aware projection operations plus one intersection type fusion operation. □

Type-aware union operation: Similar to the type-aware intersection operation, on the entity structure information embedding module, we perform one entity union operation on it. In the entity type information embedding module, we separately record different path type compatibility and add the paths that need to be united to the union path list and the relevant information in the computation order list. We deal with it during the final path fusion meta-operation.

Lemma 3. *In the logical patterns discussed in the experiments of this paper, the above solution is equivalent to Problem 3. (We prove this with a simple example: given a three-union query, the anchor entity set E contains e_1, e_2, e_3, the target entity is called e_4, and the relation is a list R with r_1, r_2, r_3.)*

Proof. According to the structural projection operation, we have:

$$\mathbb{U}(E) \Leftrightarrow \cup_{i=1}^{3}\mathbb{P}(e_i, r_i); \tag{21}$$

we can prove this similarly to the intersection operation:

$$Fu(e_4, R, \mathbb{U}_t(E, C)) \Leftrightarrow Fu(e_4, R, \mathbb{U}(E) \cup (\mathbb{P}_t(e_1, r_1) \cup \mathbb{P}_t(e_2, r_2) \cup \mathbb{P}_t(e_3, r_3))); \tag{22}$$

this shows that the scores calculated based on our algorithm are consistent with using three type-aware projection operations plus one union type fusion operation. □

We substitute Lemmas 1 and 2 into Algorithm 1, which becomes similar to the form of query embedding generation algorithm proposed in GQE [14], with the difference that UnTiCk's query embedding generation uses type-aware neural logical operators instead of those containing only structure information. Moreover, our algorithm can be extended to merge set operations. As in the two-union-projection logical pattern, it only needs to replace the intersection part with the union (i.e., substitution of Lemmas 1 and 3 into Algorithm 1) and moving the union part to the later. Because the computation graph has been converted to the DNF using the DNF-query rewriting strategy before applying the algorithm, the other logical patterns discussed in experiments can also be proven recursively by applying a form similar to Algorithm 1.

In summary, we proved the effectiveness of the proposed method on the UnTiCk problem presented in Section 3.3.

4.3. Optimization Objective

Our training goal was to jointly learn type-aware logical embedding operators and optimize entity structure embeddings and entity type embeddings in proportion. Therefore, we designed the loss function as a joint form and controlled the ratio of the two modules with a hyperparameter, combining both entity query operations and unsupervised type operations. We optimize our loss function to make the loss as small as possible below:

$$Loss = L_1 + \mu L_2 \tag{23}$$

where L_1 and L_2 are the loss functions of the entity structure information embedding module and the entity type information embedding module, respectively. We used type constraint weight μ as a hyperparameter so that the model does not overly rely on unsupervised type constraints and ignores entity structure information. For both modules, we used the negative sampling loss [40], which is widely used in such tasks. The form is the following equation:

$$L_1 = -log\sigma(\gamma_1 - S_1(\mathbf{e}, \mathbf{q})) - \sum_{i=1}^{k}\frac{1}{k}log\sigma(S_1(\mathbf{e_i'}, \mathbf{q}) - \gamma_1), \tag{24}$$

$$L_2 = -log\sigma(\gamma_2 - S_2(\mathbf{t_e}, \mathbf{W}, \mathbf{c})) - \sum_{i=1}^{k}\frac{1}{k}log\sigma(S_2(\mathbf{t'_{e_i}}, \mathbf{W}, \mathbf{c})) - \gamma_2), \tag{25}$$

where γ_1 and γ_2 represent the hyperparameter about the fixed margin of entity structure information embedding and entity type information embedding, respectively, S_1 and S_2 represent their score functions, e and t_e represent their positive samples, and e_i' and t'_{e_i} represent their negative samples.

5. Experiments

To validate the efficacy of our method, we tested its performance on nine different logical query patterns using three datasets widely used in the field of complex logical query embedding, with parameter settings similar to the baseline model. In this section, we describe our experiments in detail.

5.1. Datasets

We used three widely used publicly available datasets for our experiments:

- **FB15k:** FB15k [4] is a subset created from Freebase and is a frequently used standard dataset for embedding knowledge graphs. It includes knowledge base relation triples and textual references to Freebase entity pairs.
- **FB15k-237:** FB15k-237 [41] is a variant of the original FB15k dataset in which inverse relations have been eliminated because it was discovered that inverting triplets in the training set yielded many test triplets, which may cause test leakage.
- **NELL-995:** NELL [42] is a dataset built from the web via an intelligent agent named Never-Ending Language Learner. NELL-995 [11] is a subset of NELL that filters and improves NELL to make it more suitable for multi-hop reasoning tasks.

In contrast to other articles in the field, we added three additional datasets to demonstrate the model's performance on a wider range of data features:

- **WN18:** WN18 [4] is a dataset commonly used for knowledge graph linkage prediction, deriving its name from it as a subset of WordNet containing 18 different relations. Its entities correspond to senses, and the relation types define the lexical relations between senses.
- **WN18RR:** WN18RR [43] is a subset created from WN18 in order to handle test leakage due to training set triplet inversion in WN18.
- **YAGO3-10:** YAGO3-10 is a publicly available and commonly used dataset, which is a subset of YAGO3 [44], which only contains entities with at least ten relations. Most triples are descriptive attributes of people. We processed it to fit our experiments, as described below.

The details of the datasets are shown in Table 2. For FB15k, FB15-237, and NELL-995, we used the same data as Query2Box. For WN18 and WN18RR, we followed the training set, validation set, and test set division from the original paper. Since the set division of YAGO3-10 was skewed for our task, we randomly selected 5 percent of its triples from the training set. Then, we removed the triples in the validation and test sets containing entities not present in the training set. For the above division of three additional datasets, we finally applied the complex logical query generation process consistent with Query2Box to generate logical query datasets. Every KG was augmented with inverse relations consistent with the baseline, allowing our model to capture the type constraints better. The training knowledge graph is a subset of the validation knowledge graph, and the validation knowledge graph is a subset of the testing knowledge graph. Then, different logical query patterns are generated by pre-order traversal. In the end, we filter out the answers that can be fully answered by the previous smaller knowledge graph and focus only on the non-trivial answers (the answer appears in the test set, but never in the training and validation set).

The experiments were conducted on the following nine logical query patterns, containing five logical query patterns that appear in the training set and four promoted logical query patterns that appear only in the validation and test sets, as shown in Figure 4.

Table 2. Datasets' details.

Items	Statistics		Training			Validation			Test		
Dataset	Entities	Relations	Single [1]	Complex [2]	Triples	Single	Complex	Triples	Single	Complex	Triples
FB15k	14,951	1345	273,710	273,710	483,142	59,097	8000	50,000	67,016	8000	59,071
FB15k-237	14,505	237	149,689	149,689	272,115	20,101	5000	17,526	22,812	5000	20,438
NELL-995	63,361	200	107,982	107,982	114,213	16,927	4000	14,324	17,034	4000	14,267
WN18	40,943	18	171,254	171,254	141,442	9006	3000	5000	9028	3000	5000
WN18RR	40,943	11	103,509	103,509	86,835	5202	2000	3034	5356	2000	3034
YAGO3-10	51,374	36	64,420	40,000	53,554	3998	1500	2250	4160	1500	2333

[1] Single: 1p logical query pattern. [2] Complex: other logical query patterns.

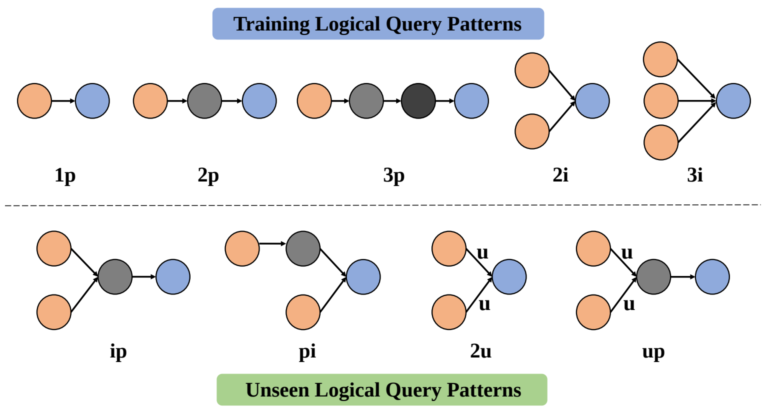
Training Logical Query Patterns
1p
2p
3p
2i
3i
u
u
u
u
ip
pi
2u
up
Unseen Logical Query Patterns

Figure 4. This is a diagram of the nine logical query patterns. It contains five training logical query patterns and four unseen logical query patterns. **p** represents *projection*, **i** *intersection*, and **u** *union*.

5.2. Evaluation Metrics

We used the common metrics for evaluating this type of task. Given a test query q, we calculated the score using the formula $E_1 + \mu E_2$ to sort the non-trivial answers in our query, and we used the sorted *Mean Reciprocal Rank (MRR)* and *Hits at K (H@K)*.

$$MRR = \frac{1}{|[\![q^t]\!]|} \sum_{e \in [\![q^t]\!]}^{|[\![q^t]\!]|} \frac{1}{Rank(e)} \tag{26}$$

where $[\![q^t]\!]$ is the non-trivial answers' set, $Rank(e)$ is a function that calculates the rank of an entity after a scoring function, and e is for the entity that has been scored.

$$Hit@K = \mathbb{1}[Rank(e) \leq K], \tag{27}$$

$$\mathbb{1}[r \leq K] \begin{cases} 1 & r \leq K \\ 0 & otherwise \end{cases}. \tag{28}$$

MRR calculates the average of the reciprocal of answer entities' *rank*, and *Hit@K* is the proportion of queries with *rank* less than or equal to K.

5.3. Baselines and Hyperparameter Settings

In our experiments, we selected two representative models of the logical query embedding domain and set up three baselines:

1. **GQE** [14] treats entities as points and queries as individual vectors. We used its extended version. The projection is modeled as a translation mechanism and the intersection as Deepsets [38], and then, the DNF-query rewriting strategy is used to make it support union operations;

2. **GQE-Double**, which has the same basic model settings as GQE, but uses double-embedding dimensionality to enable the model dimensionality to be on the same level as other models;
3. **Query2Box** [21] models the query as a box embedding, projection as a linear transformation, intersection as the center using the attention mechanism, offset using Deepsets, and the sigmoid function to shrink, and union uses the same DNF-query rewriting strategy.

For the three baselines above, we followed the main default parameters of the implementation (https://github.com/hyren/query2box, accessed on 17 January 2023). The purpose was to fairly compare and control the variables, which can test our model's improvement without affecting the plug-in module's performance. For GQE, we used 400 embedding dimensionality. GQE-Double used 800 embedding dimensionality. Query2Box used 400 embedding dimensionality (center and offset, respectively). All models were trained with margin $\gamma_1 = 24$, a learning rate of 0.0001, a batch size of 512, and 128 negative samples. Because of the characteristics of the different datasets, we trained each model for 300,000 iterations in the performance experiments and 120,000 iterations in the robustness experiments.

Based on this, we implemented three different combinations of UnTiCk. For the entity structure information embedding part, we used the same hyperparameter settings as above, except that the embedding dimensionality became 360, 760, and 360. For the entity type constraint embedding part, we used the embedding dimensionality of 40. They all used the type margin of $\gamma_2 = 8$ and the type constraint weight of $\mu = 0.3$. Such a setting ensured a consistent embedding dimensionality. We trained each model for 300,000 iterations and 120,000 iterations as above.

We also tried to combine with NewLook [22], but ultimately did not consider this part of the work due to GPU limitations, since NewLook is based on the structure of Query2Box for further work. The experimental results of Query2Box can represent, to some extent, the effective prospect of our approach to this family of models (box embedding-based model).

5.4. Experimental Results and Discussion

We conducted performance experiments relying on baselines on the popular three datasets. For the result of Hit@3, a commonly used and balanced metric in the tasks, we show the experimental results in Table 3. Our model outperformed the baseline model on the average of all nine logical query patterns.

Table 3. Performance experimental results of Hit@3. The best results are indicated in bold.

Datasets	Models	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
FB15k	GQE	0.3979	0.6448	0.3508	0.2543	0.5434	0.6567	0.1491	0.3191	0.3865	0.2764
	GQE-Double	0.4058	0.6498	0.3578	0.2596	0.5566	0.6732	0.1564	0.3349	0.3828	0.2812
	Query2Box	0.4872	0.7883	0.4175	0.3087	0.5908	0.7119	0.2111	0.4124	0.6122	0.3317
	UnTiCk (GQE)	0.4484	0.6924	0.4071	0.3191	0.5843	0.6869	0.1846	0.3759	0.4576	0.3275
	UnTiCk (GQE-D)	0.4577	0.6977	0.4124	0.3287	0.6016	0.7068	0.1937	0.3909	0.4571	0.3300
	UnTiCk (Q2B)	**0.5010**	**0.7913**	**0.4420**	**0.3547**	**0.6178**	**0.7310**	**0.2286**	**0.4413**	**0.6199**	**0.3631**
FB15k-237	GQE	0.2305	0.4044	0.2141	0.1557	0.2993	0.4179	0.0859	0.1728	0.1634	0.1613
	GQE-Double	0.2388	0.4100	0.2190	0.1577	0.3206	0.4374	0.0877	0.1851	0.1662	0.1656
	Query2Box	0.2702	0.4692	0.2504	0.1893	**0.3208**	**0.4486**	0.1091	**0.2087**	0.2453	0.1902
	UnTiCk (GQE)	0.2473	0.4286	0.2465	0.1932	0.2829	0.3999	0.0961	0.1795	0.2080	0.1914
	UnTiCk (GQE-D)	0.2565	0.4378	0.2507	0.1943	0.3049	0.4258	0.1006	0.1906	0.2120	0.1922
	UnTiCk (Q2B)	**0.2753**	**0.4715**	**0.2619**	**0.2082**	0.3056	0.4471	**0.1122**	0.2061	**0.2562**	**0.2087**
NELL-995	GQE	0.2514	0.4262	0.2295	0.2060	0.3205	0.4585	0.0788	0.1840	0.2120	0.1468
	GQE-Double	0.2588	0.4282	0.2368	0.2110	0.3369	0.4821	0.0814	0.1930	0.2113	0.1481
	Query2Box	0.3078	**0.5549**	0.2652	0.2354	**0.3492**	**0.4822**	0.1328	0.2113	0.3695	0.1693
	UnTiCk (GQE)	0.2770	0.4165	0.2685	0.2659	0.3205	0.4585	0.0910	0.1972	0.2745	0.2000
	UnTiCk (GQE-D)	0.2823	0.4166	0.2793	0.2731	0.3286	0.4678	0.0948	0.2020	0.2684	**0.2104**
	UnTiCk (Q2B)	**0.3189**	0.5457	**0.2936**	**0.2800**	0.3431	0.4804	**0.1349**	**0.2123**	**0.3781**	0.2017

Specifically, on the FB15k dataset, our model outperformed all baselines on the five logical query patterns that appear in the training set and on the four unseen logical query patterns. On FB15k-237, our model was slightly lower than the baseline model on the 2i, 3i, and pi logical query patterns and better than the baseline on all other logical query patterns. On NELL, our model was slightly lower than the original model on the 1p, 2i, and 3i logical query patterns and better than the baseline on all unseen logical patterns. The results demonstrate the effectiveness of our model in reasoning about the correct answer.

The MRR results in Table 4 were similar to the results above. This demonstrates the excellent modularity and generalization of our model. The results of Hit@1 and Hit@10 shown in Tables 5 and 6 are generally consistent with the above analysis, and we do not go into detail here.

Table 4. Performance experimental results of MRR. The best results are indicated in bold.

Datasets	Models	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
FB15k	GQE	0.3371	0.5114	0.3056	0.2241	0.4614	0.5626	0.1375	0.2769	0.3066	0.2479
	GQE-Double	0.3440	0.5067	0.3113	0.2274	0.4775	0.5840	0.1449	0.2910	0.3024	0.2509
	Query2Box	0.4153	0.6604	0.3795	0.2778	0.4934	0.6021	0.1933	0.3499	0.4762	0.3053
	UnTiCk (GQE)	0.3909	0.5816	0.3721	0.2933	0.5025	0.6030	0.1703	0.3258	0.3697	0.3000
	UnTiCk (GQE-D)	0.3987	0.5764	0.3773	0.2991	0.5201	0.6239	0.1782	0.3420	0.3649	0.3065
	UnTiCk (Q2B)	**0.4396**	**0.6760**	**0.4048**	**0.3208**	**0.5209**	**0.6266**	**0.2081**	**0.3775**	**0.4882**	**0.3334**
FB15k-237	GQE	0.2047	0.3469	0.1941	0.1430	0.2566	0.3631	0.0850	0.1583	0.1441	0.1509
	GQE-Double	0.2127	0.3503	0.1965	0.1477	**0.2776**	**0.3860**	0.0893	0.1690	0.1467	0.1512
	Query2Box	0.2369	0.4033	0.2276	0.1760	0.2737	0.3769	0.1060	**0.1847**	0.2050	0.1793
	UnTiCk (GQE)	0.2218	0.3754	0.2242	0.1794	0.2458	0.3541	0.0953	0.1663	0.1768	0.1751
	UnTiCk (GQE-D)	0.2286	0.3772	0.2279	0.1815	0.2657	0.3774	0.0973	0.1748	0.1766	0.1792
	UnTiCk (Q2B)	**0.2414**	**0.4100**	**0.2397**	**0.1917**	0.2654	0.3765	**0.1061**	0.1821	**0.2080**	**0.1935**
NELL-995	GQE	0.2133	0.3161	0.1952	0.1769	0.2799	0.4072	0.0780	0.1667	0.1647	0.1349
	GQE-Double	0.2188	0.3189	0.1999	0.1797	0.2901	**0.4274**	0.0791	0.1744	0.1643	0.1352
	Query2Box	0.2560	**0.4145**	0.2297	0.2106	**0.2915**	0.4183	0.1251	0.1918	0.2657	0.1566
	UnTiCk (GQE)	0.2364	0.3228	0.2329	0.2371	0.2766	0.4039	0.0871	0.1815	0.2073	0.1781
	UnTiCk (GQE-D)	0.2419	0.3253	0.2409	0.2408	0.2856	0.4157	0.0899	0.1846	0.2110	**0.1837**
	UnTiCk (Q2B)	**0.2658**	0.4106	**0.2494**	**0.2513**	0.2879	0.4156	**0.1253**	**0.1935**	**0.2766**	0.1816

Table 5. Performance experimental results of Hit@1. The best results are indicated in bold.

Datasets	Models	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
FB15k	GQE	0.2185	0.3310	0.2009	0.1400	0.3213	0.4214	0.0786	0.1690	0.1492	0.1554
	GQE-Double	0.2234	0.3171	0.2032	0.1391	0.3392	0.4499	0.0826	0.1829	0.1411	0.1555
	Query2Box	0.2904	0.5043	0.2790	0.1867	0.3442	0.4553	0.1183	0.2271	0.2879	0.2107
	UnTiCk (GQE)	0.2758	0.4291	0.2759	0.2039	0.3696	0.4753	0.1006	0.2097	0.2116	0.2068
	UnTiCk (GQE-D)	0.2829	0.4147	0.2804	0.2077	**0.3895**	**0.5012**	0.1072	0.2285	0.2029	0.2136
	UnTiCk (Q2B)	**0.3175**	**0.5326**	**0.3068**	**0.2235**	0.3762	0.4871	**0.1316**	**0.2537**	**0.3095**	**0.2367**
FB15k-237	GQE	0.1203	0.2277	0.1196	0.0818	0.1433	0.2455	0.0430	0.0868	0.0540	0.0806
	GQE-Double	0.1277	0.2295	0.1198	0.0861	**0.1653**	**0.2719**	0.0472	0.0963	0.0543	0.0785
	Query2Box	0.1432	0.2791	0.1457	0.1077	0.1541	0.2472	**0.0559**	0.1025	0.0917	0.1049
	UnTiCk (GQE)	0.1354	0.2614	0.1447	0.1104	0.1403	0.2418	0.0500	0.0944	0.0771	0.0988
	UnTiCk (GQE-D)	0.1415	0.2587	0.1467	0.1114	0.1595	0.2681	0.0503	0.1027	0.0730	0.1032
	UnTiCk (Q2B)	**0.1495**	**0.2903**	**0.1604**	**0.1206**	0.1467	0.2477	0.0554	**0.1051**	**0.1017**	**0.1176**
NELL-995	GQE	0.1140	0.1481	0.1026	0.0952	0.1598	0.2852	0.0363	0.0960	0.0436	0.0594
	GQE-Double	0.1186	0.1513	0.1055	0.0963	0.1678	**0.3082**	0.0363	0.1012	0.0435	0.0569
	Query2Box	0.1466	0.2309	0.1336	0.1298	0.1655	0.2883	**0.0727**	0.1164	0.1036	0.0786
	UnTiCk (GQE)	0.1348	0.1719	0.1371	0.1492	0.1587	0.2850	0.0400	0.1090	0.0696	0.0925
	UnTiCk (GQE-D)	0.1404	0.1742	0.1469	0.1520	**0.1679**	0.2990	0.0408	0.1115	0.0740	0.0970
	UnTiCk (Q2B)	**0.1563**	**0.2329**	**0.1522**	**0.1671**	0.1651	0.2882	0.0712	**0.1177**	**0.1147**	**0.0972**

From the performance experimental results, the patterns with lower comparative baselines for the models on FB15k-237 and NELL are mainly focused on the logical query patterns that require type feature fusion meta-operations. Since it performs well on all patterns of FB15k, we think that this is due to a certain degree of overfitting caused by the fewer different relations in FB15k-237 and NELL, as well as the overly rich constraints (multiple logical path joint constraints) on the logical query patterns. The results illustrated, from another perspective, that the number of distinct relations in the dataset was positively correlated with our model's performance.

To test the robustness of our model on different feature and size datasets, we produced three additional datasets from WN18, WN8RR, and YAGO3-10. They have commonly used datasets on link prediction, and we made versions adapted to complex logical query tasks.

Many patterns on these datasets lose reality due to the entity-relation ratio and the dataset division. We used each pattern's relative performance instead of the absolute performance as an evaluation criterion. Our purpose was, compared to the complex logical query embedding models, to test the robustness of our model under different data characteristics. We draw stacked column charts of the Hit@3 results on the three datasets in Figure 5.

Table 6. Performance experimental results of Hit@10. The best results are indicated in bold.

Datasets	Models	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
FB15k	GQE	0.5574	0.8136	0.5046	0.3825	0.7212	0.8151	0.2485	0.4876	0.6125	0.4314
	GQE-Double	0.5687	0.8220	0.5142	0.3901	0.7362	0.8276	0.2643	0.5034	0.6196	0.4410
	Query2Box	0.6422	**0.9060**	0.5799	0.4536	0.7564	0.8518	0.3384	0.5812	**0.8112**	0.5013
	UnTiCk (GQE)	0.6090	0.8379	0.5621	0.4752	0.7474	0.8371	0.3039	0.5531	0.6749	0.4896
	UnTiCk (GQE-D)	0.6173	0.8425	0.5729	0.4809	0.7586	0.8472	0.3137	0.5669	0.6723	0.5009
	UnTiCk (Q2B)	**0.6611**	0.9013	**0.6042**	**0.5096**	**0.7748**	**0.8646**	**0.3544**	**0.6108**	0.7993	**0.5312**
FB15k-237	GQE	0.3711	0.5741	0.3349	0.2625	0.4881	0.5976	0.1618	0.2965	0.3337	0.2909
	GQE-Double	0.3802	0.5773	0.3428	0.2704	0.5047	0.6140	0.1661	0.3101	0.3417	0.2950
	Query2Box	0.4207	0.6402	0.3887	0.3128	**0.5119**	**0.6225**	0.2014	**0.3432**	0.4396	0.3257
	UnTiCk (GQE)	0.3921	0.5945	0.3824	0.3192	0.4593	0.5758	0.1801	0.3062	0.3837	0.3275
	UnTiCk (GQE-D)	0.4004	0.6023	0.3869	0.3228	0.4826	0.5906	0.1843	0.3147	0.3871	0.3327
	UnTiCk (Q2B)	**0.4251**	**0.6410**	**0.4042**	**0.3399**	0.4925	0.6161	**0.2016**	0.3362	**0.4413**	**0.3527**
NELL-995	GQE	0.4076	0.6047	0.3767	0.3252	0.5386	0.6546	0.1526	0.3044	0.4151	0.2967
	GQE-Double	0.4151	0.6111	0.3829	0.3341	0.5461	0.6666	0.1581	0.3151	0.4190	0.3033
	Query2Box	0.4673	**0.7116**	0.4244	0.3652	**0.5534**	**0.6757**	0.2265	**0.3417**	0.5784	0.3289
	UnTiCk (GQE)	0.4392	0.5912	0.4313	0.4095	0.5219	0.6429	0.1762	0.3239	0.4903	**0.3660**
	UnTiCk (GQE-D)	0.4426	0.5915	0.4331	0.4088	0.5280	0.6502	0.1845	0.3263	0.4955	0.3659
	UnTiCk (Q2B)	**0.4765**	0.6983	**0.4419**	**0.4176**	0.5448	0.6686	**0.2275**	0.3386	**0.5913**	0.3600

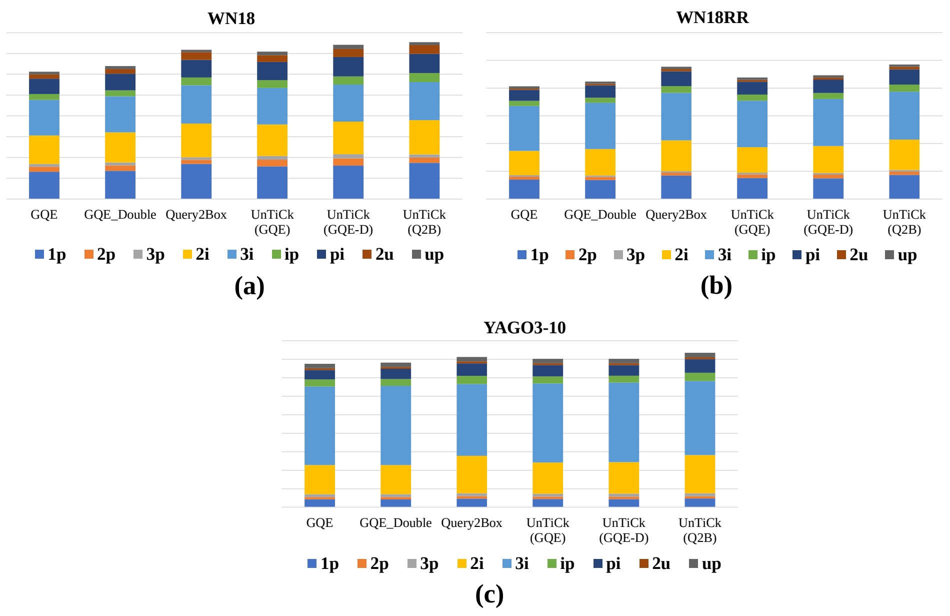

Figure 5. Stacked column chart of robustness of Hit@3 results on additional datasets: (**a**) Stacked Hit@3 results on WN18. (**b**) Stacked Hit@3 results on WN18RR. (**c**) Stacked Hit@3 results on YAGO3-10.

With the results of the robustness experiments, our model still consistently outperformed the baseline model on more different data features. Although we do not refer to the significance of the absolute performance on these three datasets, it can still be seen from the data that the model performance improvement on YAGO3-10 was less compared to WN18 and WN18RR. Since the triples on WN18 and WN18RR represent character relations between senses, their logical patterns can somewhat match the characteristics of real logical problems. For YAGO3-10, the logical patterns that may exist in them lose their real meaning due to the numerous relations of each entity, but the limited overlap of entities. Random sampling further adds to this feature. This also brings about a sparse distribution of entities in the embedding space, a more significant differentiation of entities, and fewer answers in the result set that do not conform to the type constraints. In other words, our model was more effective on datasets having evenly distributed type features and moderate entity-relation ratio.

Moreover, our model prefers to increase the fine-grained differentiation in broad fuzzy categories. The optimization space for datasets with a sparse type space is poor, leading to less model improvement. This feature will be confirmed in the visualization results later on. However, in most cases, our method gave better results, demonstrating the robustness of our approach.

To verify the effectiveness of our embedding dimensionality setting, we used different parameters on UnTiCk (GQE-Double). For example, UnTiCk (GD-760-40) represents UnTiCk (GQE-Double) with an entity structure information embedding dimensionality of 760 and an entity type information embedding dimensionality of 40. We used this setting as the standard parameter in the previous experiment. We used the setting of the entity type information embedding dimension sequence [0, 20, 40, 60, 80] and set the entity structure information embedding dimension sequence as [800, 780, 760, 740, 720] for the performance experiments. We used the Hit@3 results on three experimental performance datasets. The clustered column charts and stacked bar chart of the Hit@3 results are shown in Figure 6. In the clustered column charts, we highlight our models of the default parameters using the black line plots.

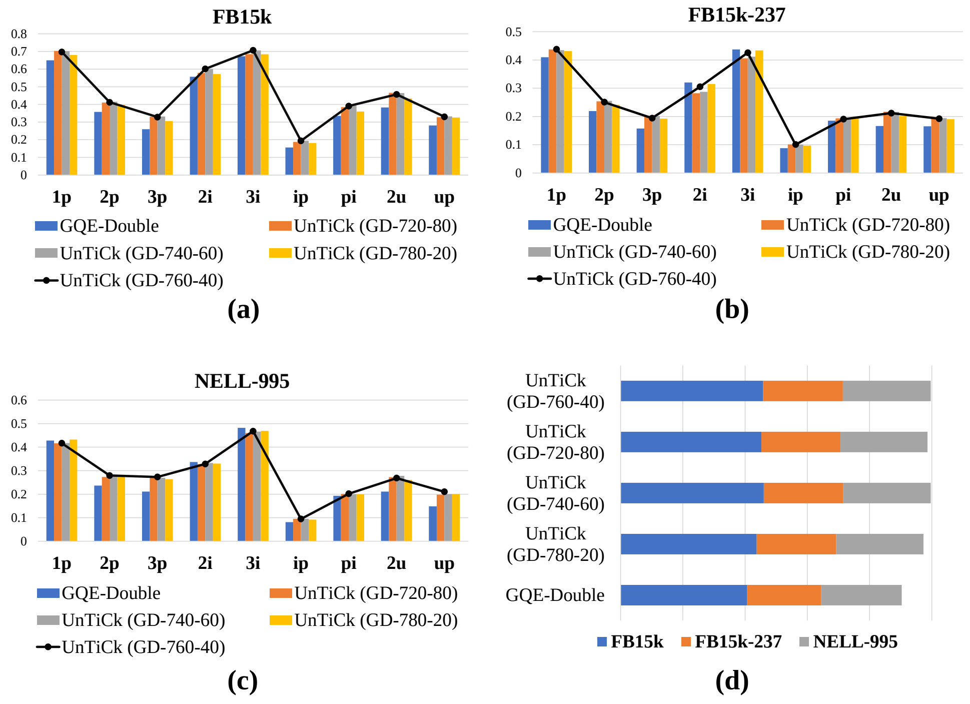

Figure 6. The clustered column charts and stacked bar chart of the Hit@3 results for different parameters of UnTiCk (GQE-Double): (**a**) Hit@3 results for different parameter on FB15k. (**b**) Hit@3 results for different parameter on FB15k-237. (**c**) Hit@3 results for different parameters on NELL-995. (**d**) stacked Hit@3 results for different parameters on three datasets.

We found a steady improvement for all our models in the interval of entity type information embedding dimensionality as above. There was a peak in the overall effectiveness of the model, achieving its best results at 40 to 60. As the type embedding dimension continues to increase or decrease, the overall effectiveness of the model starts to decrease. This confirmed our previous statement that a good ratio between entity structure information embedding and type information embedding needs to be maintained to capture and utilize both of the above fully. In our experience, this ratio is generally close to 10:1 for the box-based model and 20:1 for the point-based model.

We produced the visualization of the experimental results. The data processing used a similar approach to TypeDM and TypeComplex [19] for data processing. In detail, we used the dataset of display types already labeled on FB15k provided by TKRL [15], which labeled various relevant types for each entity. From these, we filtered out entities containing five common categories, including *people*, *location*, *organization*, *film*, and *sports*. Each entity was assigned to only one category.

Then, we used t-SNE [45] for dimensionality reduction to two dimensions for the weights obtained from model training, labeled the above five types of entities with different colors, and finally, visualized the results. Here, we used the type constraint embeddings of UnTiCk (Q2B), the entity structure information embedding of UnTiCk (Q2B), and the entity information embedding of the original Query2Box. The visualization results are shown in Figure 7.

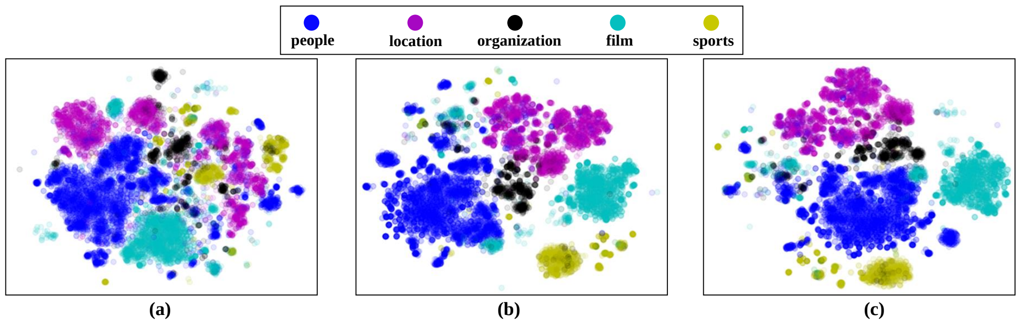

Figure 7. Visualization of the embedding of UnTiCk (Q2B) and original Query2Box: (**a**) Result of entity type constraint embedding of UnTiCk (Q2B). (**b**) Result of entity structure information embedding of UnTiCk (Q2B). (**c**) Result of entity information embedding of the original Query2Box.

We can see that the entity type constraint embedding of UnTiCk tends to produce smaller clusters of the same type. It represents a more fine-grained information extraction in dealing with the diversity of entity types. Since the entity structure information embedding part of UnTiCk adopts a similar approach to the popular complex logical query embedding model, there is no significant difference in the visualization results of the two parts.

Entity structure information tends to divide the whole graph into chunks of several types. The visualization results prove that, although it allows structurally similar entities to exhibit some primary type features to some extent, it cannot obtain more fine-grained and precise unsupervised type features.

For distance-based models [9] like the embedding model, this characteristic makes the clusters of entities in the space exhibit more unsupervised type constraints. In other words, this data distribution makes the entities close to the target entities conform more to the type constraints compared to models that are also box or point modeled and based only on entity structure information.

We further demonstrate the effectiveness of our approach with another visualization result. Here, we filtered out the entities of *people* and treated all the remaining entities as *others*. We chose the relation */people/person/profession*, which frequently occurs in *people*. We compared the original entity type embedding of *people* with the entity type embedding after the type feature extraction meta-operation in the type constraint embedding module. The visualization is shown in Figure 8.

It is obvious that, although the entities of *people* show smaller clusters in the embeddings, when we performed the type feature extraction meta-operation, the obtained relation-specific type representations became large clusters and more accurately separate *people* and *others* at this time. The visualization results not only demonstrate our approach's effectiveness for capturing the diversity of entity types, but also prove that our approach can work well for unsupervised entity type constraints.

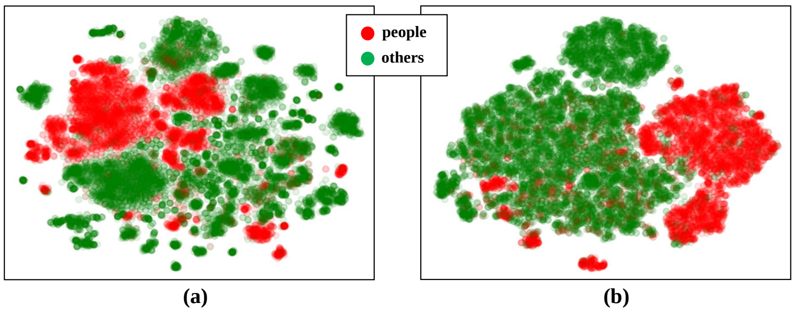

Figure 8. Visualization about type *people* and relation */people/person/profession*: (**a**) Result without type feature extraction meta-operation. (**b**) Result after using type feature extraction meta-operation.

To illustrate the difference between the proposed model and other models, we used the trained model for two cases to show its results. Here, a 3*c* query and a *uc* query were chosen because the type of answer space is rich (the intersection operation will shrink the answer space). They were used as examples under the single path and required path fusion. The specific query content and model prediction results are shown in Table 7.

Table 7. Case study for UnTiCk (Q2B) and Query2Box.

Anchor Entities	Relation	Target Entities	Query2Box Predication	UnTiCk (Q2B) Predication
The Last King of Scotland (film)	(1) films_in_this_genre_reverse, (2) titles, (3) production_companies	(1) Channel 4 (organization), (2) SONY (organization)	(1) Walt Disney Animation Studios (organization), (2) Walt Disney Studios Motion Pictures (organization), (3) drama film (film_genre), (4) Pixar (organization), (5) historical drama (film_genre)	(1) Walt Disney Animation Studios (organization), (2) Magnolia Pictures (organization), (3) Walt Disney Studios MotionPictures (organization), (4) Pixar (organization), (5) BBC (organization)
(1) Chester County (location), (2) Latin Grammy Award for Album of the Year (award_category)	(1) contains, (2) award_reverse, (3) people_with_this_profession_reverse	(1) model (profession), (2) audio engineer (profession), (3) songwriter (profession)	(1) songwriter (profession), (2) Chester County (location), (3) composer (profession), (4) actor (profession), (5) artist (profession)	(1) songwriter (profession), (2) artist (profession), (3) actor (profession), (4) composer (profession), (5) guitarist (profession)

The first column of the table is the case study result of 3*c*; the anchor row represents the query's anchor node, and the relation list represents the three query relations in order. The second column is the case study result of *uc*; the anchor list represents the two anchor nodes of the query in order, and the relation list is the relations connected with the two entities and the relation after the union.

From the results in the table, our model was better in type consistency for the first five answers compared to Query2Box, achieving "untick" for entities that do not meet the type constraints. We also collected the results for two cases. On the 3*c* query, UnTiCk (Q2B) had a prediction rank of 39 and 102 for the two answers, gaining improvement over Query2Box's 90 and 357. On the *uc* query, this result was 8, 12, and 1, compared with 19, 25, and 1. Both cases prove our model's effectiveness from the consistency of the result types and the ranking of the results.

6. Conclusions and Future Work

In this paper, we proposed UnTiCk, which provides the ability to capture unsupervised type constraints to the traditional embedding-based complex logical query framework. We analyzed and designed type compatibility measurement meta-operations, then combined them into an existing complex logical query framework to implement type-aware neural logical operators. We innovatively solved the problem of unsupervised type constraints in complex logical queries, and we modeled the diversity of unsupervised type representations of entities in multi-hop logical paths. With experimental and visualization results on popular datasets, we steadily outperformed traditional complex logical query embedding frameworks for the same number of embedding dimensions. This demonstrated the effectiveness and robustness of our framework.

In future work, we will try to extend our unsupervised type constraint extraction method to the complete set of first-order logical operators. Another promising work is to improve the atomicity and generalization of type meta-operators by further refining the characteristics of entity type information in the complex logical query embedding process. This allows the model to be effectively utilized on a wider variety of logical query embedding models. In addition, making more datasets with different logical patterns and data characteristics is an important task to drive complex logical embedding models closer to practical applications.

Author Contributions: Conceptualization, D.C. and J.G.; methodology, D.C. and Q.L.; software, D.C.; validation, D.C. and Q.L.; formal analysis, D.C.; investigation, D.C. and Q.L.; resources, J.G.; data curation, D.C. and Q.L.; writing—original draft preparation, D.C.; writing—review and editing, D.C., Q.L., and J.G.; visualization, D.C.; supervision, J.G.; project administration, J.G.; funding acquisition, J.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program (Grant Number 2020AAA0108500), the National Natural Science Foundation of China (Grant Number U1836118), the Key Research and Development Program of Wuhan (Grant Number 2022012202015070), and the Open Research Fund of Key Laboratory of Rich Media Digital Publishing, Content Organization and Knowledge Service (Grant Number ZD2022-10/05).

Data Availability Statement: The data presented in this study are openly available in [4,15,21,43], and the public repository links are https://github.com/hyren/query2box, https://everest.hds.utc.fr/doku.php?id=en:transe, https://github.com/TimDettmers/ConvE, and https://github.com/thunlp/TKRL, accessed on 17 January 2023.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J. Freebase: A collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 10–12 June 2008; pp. 1247–1250.
2. Suchanek, F.M.; Kasneci, G.; Weikum, G. YAGO: A core of semantic knowledge. In Proceedings of the 16th International Conference on World Wide Web, Alberta, AB, Canada, 8–12 May 2007; pp. 697–706.
3. Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z. DBpedia: A nucleus for a web of open data. In *The Semantic Web*; Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., et al. Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 722–735.
4. Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; Yakhnenko, O. Translating embeddings for modeling multi-relational data. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013.
5. Ji, G.; He, S.; Xu, L.; Liu, K.; Zhao, J. Knowledge graph embedding via dynamic mapping matrix. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, 26–31 July 2015; pp. 687–696.
6. Nickel, M.; Tresp, V.; Kriegel, H.P. A three-way model for collective learning on multi-relational data. In Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, 28 June–2 July 2011.
7. Yang, B.; Yih, S.W.t.; He, X.; Gao, J.; Deng, L. Embedding entities and relations for learning and inference in knowledge bases. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
8. Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; Bouchard, G. Complex embeddings for simple link prediction. In Proceedings of the International Conference on Machine Learning, New York City, NY, USA, 19–24 June 2016; pp. 2071–2080.
9. Sun, Z.; Deng, Z.H.; Nie, J.Y.; Tang, J. RotatE: Knowledge graph embedding by relational rotation in complex space. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
10. Guo, S.; Wang, Q.; Wang, L.; Wang, B.; Guo, L. Jointly embedding knowledge graphs and logical rules. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016; pp. 192–202.
11. Xiong, W.; Hoang, T.; Wang, W.Y. DeepPath: A reinforcement learning method for knowledge graph reasoning. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, 7–11 September 2017; pp. 564–573.

12. Guo, S.; Wang, Q.; Wang, L.; Wang, B.; Guo, L. Knowledge graph embedding with iterative guidance from soft rules. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
13. Wang, H.; Ren, H.; Leskovec, J. Relational message passing for knowledge graph completion. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual, 14–18 August 2021; pp. 1697–1707.
14. Hamilton, W.; Bajaj, P.; Zitnik, M.; Jurafsky, D.; Leskovec, J. Embedding logical queries on knowledge graphs. In Proceedings of the 32th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 2–8 December 2018.
15. Xie, R.; Liu, Z.; Sun, M. Representation learning of knowledge graphs with hierarchical types. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York City, NY, USA, 9–15 July 2016; pp. 2965–2971.
16. Zhao, Y.; Zhang, A.; Xie, R.; Liu, K.; Wang, X. Connecting embeddings for knowledge graph entity typing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Virtual, 5–10 July 2020; pp. 6419–6428.
17. Lu, Y.; Ichise, R. ProtoE: Enhancing knowledge graph completion models with unsupervised type representation learning. *Information* **2022**, *13*, 354:1–354:25. [CrossRef]
18. Niu, G.; Li, B.; Zhang, Y.; Pu, S.; Li, J. AutoETER: Automated entity type representation for knowledge graph embedding. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Virtual, 16–20 November 2020; pp. 1172–1181.
19. Jain, P.; Kumar, P.; Chakrabarti, S. Type-sensitive knowledge base inference without explicit type supervision. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Melbourne, Australia, 15–20 July 2018; pp. 75–80.
20. Lu, Y.; Ichise, R. Unsupervised type constraint inference in bilinear knowledge graph completion models. In Proceedings of the 2021 IEEE International Conference on Big Knowledge (ICBK), Auckland, New Zealand, 7–8 December 2021; pp. 15–22.
21. Ren, H.; Hu, W.; Leskovec, J. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In Proceedings of the 8th International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
22. Liu, L.; Du, B.; Ji, H.; Zhai, C.; Tong, H. Neural-answering logical queries on knowledge graphs. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual, 14–18 August 2021; pp. 1087–1097.
23. Ren, H.; Leskovec, J. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In Proceedings of the the 34th International Conference on Neural Information Processing Systems, Virtual, 6–12 December 2020.
24. Hu, Z.; Gutiérrez-Basulto, V.; Xiang, Z.; Li, X.; Li, R.; Pan, J.Z. Type-aware embeddings for multi-hop reasoning over knowledge graphs. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, Vienna, Austria, 23–29 July 2022; pp. 3078–3084.
25. Wang, M.; Wang, R.; Liu, J.; Chen, Y.; Zhang, L.; Qi, G. Towards empty answers in SPARQL: Approximating querying with RDF embedding. In Proceedings of the International Semantic Web Conference, Monterey, CA, USA, 8–12 October 2018; pp. 513–529.
26. Wang, Y.; Khan, A.; Wu, T.; Jin, J.; Yan, H. Semantic guided and response times bounded top-k similarity search over knowledge graphs. In Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 445–456.
27. Wang, Y.; Xu, X.; Hong, Q.; Jin, J.; Wu, T. Top-k star queries on knowledge graphs through semantic-aware bounding match scores. *Knowl. Based Syst.* **2021**, *213*, 106655. [CrossRef]
28. Dalvi, N.N.; Suciu, D. Efficient query evaluation on probabilistic databases. *VLDB J.* **2007**, *16*, 523–544. [CrossRef]
29. Zhang, Z.; Wang, J.; Chen, J.; Ji, S.; Wu, F. ConE: Cone embeddings for multi-hop reasoning over knowledge graphs. In Proceedings of the Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, Virtual, 6–14 December 2021; pp. 19172–19183.
30. Bai, J.; Wang, Z.; Zhang, H.; Song, Y. Query2Particles: Knowledge graph reasoning with particle embeddings. In Proceedings of the Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, USA, 10–15 July 2022; pp. 2703–2714.
31. Ren, H.; Dai, H.; Dai, B.; Chen, X.; Yasunaga, M.; Sun, H.; Schuurmans, D.; Leskovec, J.; Zhou, D. LEGO: Latent execution-guided reasoning for multi-hop question answering on knowledge graphs. In Proceedings of the 38th International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 8959–8970.
32. Venn, J.I. On the diagrammatic and mechanical representation of propositions and reasonings. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **1880**, *10*, 1–18. [CrossRef]
33. Ma, S.; Ding, J.; Jia, W.; Wang, K.; Guo, M. TransT: Type-based multiple embedding representations for knowledge graph completion. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Skopje, Macedonia, 18–22 September 2017; pp. 717–733.
34. Ge, X.; Wang, Y.C.; Wang, B.; Kuo, C.J. CORE: A knowledge graph entity type prediction method via complex space regression and embedding. *Pattern Recognit. Lett.* **2022**, *157*, 97–103. [CrossRef]
35. Hu, Z.; Gutiérrez-Basulto, V.; Xiang, Z.; Li, R.; Pan, J.Z. Transformer-based entity typing in knowledge graphs. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2022; pp. 5988–6001.
36. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.

37. Guu, K.; Miller, J.; Liang, P. Traversing knowledge graphs in vector space. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 318–327.
38. Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R.R.; Smola, A.J. Deep sets. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
39. Davey, B.A.; Priestley, H.A. *Introduction to Lattices and Order*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2002.
40. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. In Proceedings of the 1st International Conference on Learning Representations Workshop, Scottsdale, AZ, USA, 2–4 May 2013.
41. Toutanova, K.; Chen, D. Observed versus latent features for knowledge base and text inference. In Proceedings of the 3rd Workshop on Continuous Vector Space Models and Their Compositionality, Beijing, China, 31 July 2015; pp. 57–66.
42. Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Hruschka, E.R.; Mitchell, T.M. Toward an architecture for never-ending language learning. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010.
43. Dettmers, T.; Pasquale, M.; Pontus, S.; Riedel, S. Convolutional 2D knowledge graph embeddings. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 1811–1818.
44. Mahdisoltani, F.; Biega, J.; Suchanek, F. YAGO3: A knowledge base from multilingual wikipedias. In Proceedings of the Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 4–7 January 2015.
45. van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn Res.* **2008**, *9*, 2579–2605.