CNEQ: Incorporating numbers into Knowledge Graph Reasoning

Xianshu Peng[1], Wei Wei*[1], Kaihe Xu[2], , Dangyang Chen[2]
[1]School of Computer Science & Technology, Huazhong University of Science and Technology
[2]Ping An Property & Casualty Insurance company of China
{xspeng, weiw}@hust.edu.cn
xukaihenupt@gmail.com, chendangyang273@pingan.com.cn

Abstract

Complex logical reasoning over knowledge graphs lies at the heart of many semantic downstream applications and thus has been extensively explored in recent years. However, nearly all of them overlook the rich semantics of numerical entities (e.g., magnitude, unit, and distribution) and are simply treated as common entities, or even directly removed. It may severely hinder the performance of answering queries involving numerical comparison or numerical computation. To address this issue, we propose the **C**omplex **N**umber and **E**ntity **Q**uery model (CNEQ), which comprises a Number-Entity Predictor and an Entity Filter. The Number-Entity Predictor can independently learn the structural and semantic features of entities and numerical values, thereby enabling better prediction of entities as well as numerical values. The Entity Filter can compare or calculate numerical values to filter out entities that meet certain numerical constraints. To evaluate our model, we generated a variety of multi-hop complex logical queries including numerical values on three widely-used Knowledge Graphs: FB15K, DB15K, and YAGO15K. Experimental results demonstrate that CNEQ achieves state-of-the-art results.

1 Introduction

Complex query answering (CQA), which has emerged as a significant task in recent years, and its primary objective is to perform reasoning over knowledge graphs (KGs) and to solve complex logical queries based on the structured knowledge. In CQA, natural language queries are initially converted into a subset of first-order logic (FOL) queries, incorporating disjunctions ($\vee$), conjunctions($\wedge$), and existential($\exists$) quantifiers. For example, in Figure 1 (A), the natural language query *"Where did Canadian citizens with Turing Award graduate?"* is first converted into the corresponding logical query q, the query can be represented as directed acyclic graphs (DAGs). Although many query (or DAG) embedding-based methods can effectively solve complex queries where the answers are entities (Hamilton et al., 2018; Ren et al., 2020; Guo and Kok, 2021; Ren and Leskovec, 2020; Wang et al., 2023; Huang et al., 2024a), they still have limitations when it comes to numerical value reasoning. On one hand, numerical values' representation has always been a challenging problem in the

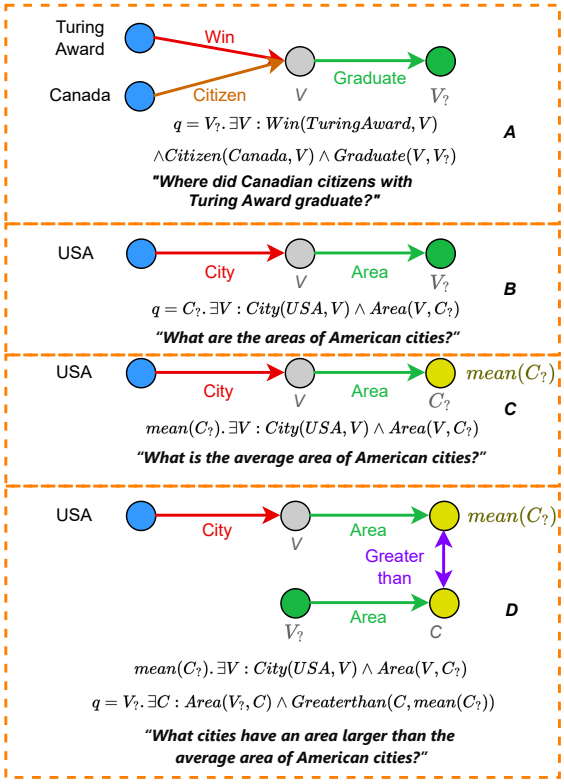

Figure 1: 4 types of complex queries: (A) represents complex queries with multi-entity answers; (B) represents complex queries with multiple numerical value answers; (C) represents complex queries with numerical statistic answers; (D) represents complex queries containing numerical constraints.

* Corresponding author.

Findings of the Association for Computational Linguistics: EACL 2024, pages 10029–10042

field of natural language processing, especially for structured knowledge graphs (García-Durán and Niepert, 2017;Thawani et al., 2021;Spithourakis and Riedel, 2018;Spokoyny and Berg-Kirkpatrick, 2020). This is because numbers are often represented as discrete nodes in KGs, which contradicts the inherent continuous nature of numbers, leading to poor reasoning performance. On the other hand, these methods overlook the features of numerical values, such as magnitude, distribution, range, and units. These features are crucial for reasoning from entities to numbers, as well as for comparing and computing between numbers. Above reasons makes these methods face challenges in complex numerical reasoning tasks.

However, complex queries involving numerical values come in various types, making the problem more difficult. For example, the query in Figure 1 (B) has answers consisting of multiple numerical values. To answer such queries, the model needs to have both multi-hop reasoning capabilities and strong modeling abilities for attribute values. Some link prediction models based on regression methods (Chung et al., 2023,Kristiadi et al., 2019,Tay et al., 2017) do consider attribute values, but they are limited as they can only perform one-hop predictions. Additionally, the query in Figure 1 (C) requires answering mean of the entire set of numerical answers. This necessitates the model to learn features such as the distribution and range of all numerical answer sets, so it is insufficient to merely learn the magnitude of individual numbers. Some complex queries also include numerical constraints, such as the query in Figure 1 (D), where "larger than" is the constraint and "the average area of American cities" is the value of the numerical constraint. To answer such complex queries, the model needs to predict statistic (eg: average area), predict individual entity attributes (eg: city's area), and compare numerical values to filter out entities that meet the constraints.

Therefore, to improve complex query tasks, we propose CNEQ, a framework designed for solving various types of queries including numerical values and entities, as illustrated in Figure 1. CNEQ comprises two modules: a Number-Entity Predictor and an Entity Filter. The Number-Entity Predictor not only uses a more reasonable method to encode continuous numerical values, thereby learning the semantic information of the numbers, such as magnitude and units. It also models the range and distribution of numerical answers sets, to capture more macro-level features. The Entity Filter can predict the attribute values of entities and filter out those that meet the conditions based on the constraints. To evaluate the performance of CNEQ in answering complex queries, we constructed a benchmark dataset for validation. Ultimately, our model achieved state-of-the-art results on the dataset. Further experimental results indicate that our approach to modeling numerical values is necessary.

Our contributions mainly include: (1) We propose a method called CNEQ to solve complex queries which explicitly models the semantic information of numerical values, and outperforms previous models in various reasoning tasks involving both numbers and entities. (2) Based on the modeling of numerical distributions, CNEQ can be used for statistics query answering. (3) We introduce a new task called "complex queries with numerical constraints" and solve such problems using CNEQ. (4) We create new benchmarks on three knowledge graphs—Freebase, Dbpedia, and YAGO—to evaluate the model's performance in answering the aforementioned types of queries.

2 Background and Preliminaries

A knowledge graph (or heterogeneous networks) can be define as $\mathcal{G} = \{(h, r, t)\} \subset (\mathcal{V} \times \mathcal{R} \times \mathcal{V}) \cup (\mathcal{V} \times \mathcal{A} \times \mathbb{R})$,where $v \in \mathcal{V}$ represents an entity node, $\mathcal{R}$ and $\mathcal{A}$ denote entity relation set and attribution set respectively. $r \in \mathcal{R} : \mathcal{V} \times \mathcal{V} \mapsto \{true, false\}$ and $a \in \mathcal{A} : \mathcal{V} \times \mathbb{R} \mapsto \{true, false\}$ can be seen as binary predicates, indicating whether the relation or attribution holds between the two elements.

For an arbitrary Existential Positive First-order (EPFO) logical queries, it can be represented by $\vee$,$\wedge$, and $\exists$. There is a concrete example in Figure 1(B), the natural language question *What are the areas of American cities?* can be represented as (1)

$$q = C_?.\exists V : City(USA, V) \wedge Area(V, C_?) \quad (1)$$

where USA represents non-variable anchor entity, V represents existentially quantified bound variable, and $C_?$ represents the target variable.

In a knowledge graph containing numerical values, We mainly discuss three types of complex queries: (1) queries where the result is multiple entities or multiple numerical values, as shown in Figure 1(A) and (B); (2) queries where the result is a statistic of all numerical answers, as shown in

Figure 1(C); (3) queries where the result is a set of entities that meet a certain numerical constraint, as shown in Figure 1(D). Next, we will formally define each type of the logical queries.

2.1 Queries with multiple entity or number answers

We define an EPFO query on incomplete knowledge graph whose answers are multiple entities or multiple numerical values as

$$q = V_? \text{ or } C_?.\ \exists V_1 \ldots V_k : e_1 \wedge e_2 \wedge \ldots \wedge e_n \quad (2)$$

where

-$e_i = r\left(v, V\right), e_i = r\left(V, V'\right)$ or $e_i = a\left(V, V'\right)$
-$V, V' \in \{V_?, C_?, V_1, \ldots, V_k\}$,
-$V \neq V', r \in \mathcal{R}, a \in \mathcal{A}, v \in \mathcal{V}$

In Equation (2), $V_?$ or $C_?$ denotes the *target variable* of the query, which means the answer can be multi-numbers or multi-entities. v is non-variable anchor entity, it indicates the entity has been mentioned in the query, $V_1 \ldots V_k$ are existentially quantified bound variables. And the answer set of query q can be represent as $[\![q]\!] \subseteq \mathcal{V} \cup \mathbb{R}$, the goal of this query is to find out all answers $v \in [\![q]\!]$ on the incomplete graph including missing answers.

2.2 Queries with statistic answers

For queries where the result is a statistic of all numerical answers,we define the EPFO query as

$$statistic(C_?).\ \exists V_1 \ldots V_k : e_1 \wedge e_2 \wedge \ldots \wedge e_n \quad (3)$$

where

-$e_i = r\left(v, V\right), e_i = r\left(V, V'\right)$ or $e_i = a\left(V, V'\right)$
-$V, V' \in \{C_?, V_1, \ldots, V_k\}$,
-$V \neq V', r \in \mathcal{R}, a \in \mathcal{A}, v \in \mathcal{V}$

In Equation (3), *statistic* is an operator which can calculate certain statistic for the whole number answer set on the incomplete graph.

2.3 Queries with numerical constraint

For queries which have numerical constrains, We divide the query into two parts, as shown in Equation (4). The first sub-query calculate the number of the numerical constraint, and the second subquery filters the entities on the knowledge graph based on the numerical constraints and the attribute values of the entities, to obtain a set of entities that meet the constraint.

$$\begin{aligned} &statistic(C_?).\ \exists V_1 \ldots V_k : e_1 \wedge e_2 \wedge \ldots \wedge e_n \\ &q = V_?.\exists C : a(V_?, C) \wedge con(C, statistic(C_?)) \end{aligned} \quad (4)$$

where

-$e_i = r\left(v, V\right), e_i = r\left(V, V'\right)$ or $e_i = a\left(V, V'\right)$
-$V, V' \in \{C_?, V_1, \ldots, V_k\}$,
-$V \neq V', r \in \mathcal{R}, a \in \mathcal{A}, v \in \mathcal{V}, C \in \mathbb{R}$

Specifically, $con : \mathbb{R} \times \mathbb{R} \mapsto \{true, false\}$ denotes an attribute filter function con_{less} (less than), $con_{greater}$ (greater than), or con_{equal} (equal to), if the two number satisfy the constrain, it return *true*.

3 Complex Number and Entity Query model

To answer complex queries that include numerical values and entities as discussed in Section 2, we propose CNEQ to tackle such complex queries, which primarily consists of two components: the Number-Entity Predictor and the Entity Filter. The Number-Entity Predictor is responsible for reasoning numerical values, entities, or statistic of certain sub-query. For example, to answer the query illustrated in Figure 2 (A), the Number-Entity Predictor is used to resolve the subquery in (1) to compute the number of numerical constraints. After that, the Entity Filter is employed to resolve the subquery in (2), filtering out the entities on the knowledge graph that satisfy the constraint. In the following parts, we will provide a detailed explanation of them.

3.1 Number-Entity Predictor

In the specific reasoning process, it involves the transformation from entity to entity or from entity to number, which introduced in Figure 2 (B). One of the Number-Entity Predictor's tasks is to model these two processes, and another task is to learn the features of numerical values. Since numerical values inherently have magnitudes, and the multiple attribute values form a distribution. Therefore, we model attribute values along two dimensions: magnitude and distribution, to prevent the loss of critical attribute information.

3.1.1 From entity to entity

For a query that involves mapping from one entity set to another entity set, such as $City(V_1, V_2)$, suppose that at step i, the embedding of the current

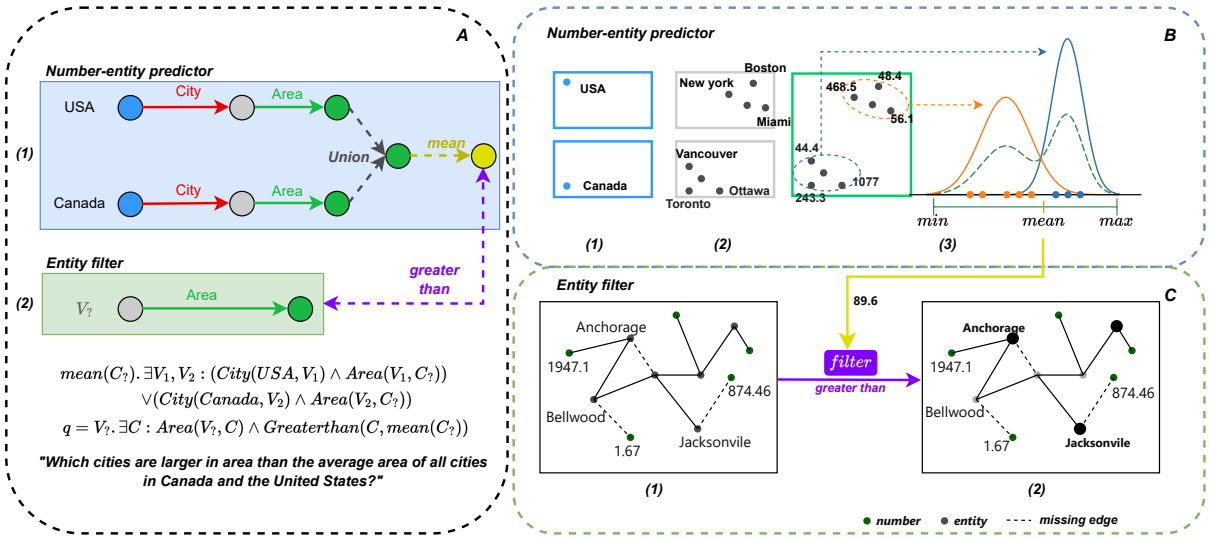

Figure 2: Illustration of our proposed CNEQ model. (A) represents the computation graph of complex queries. The Number-Entity Predictor starts from the anchor entities and iteratively execute operators to infer the statistics. The Entity Filter then find the entities that meet the numerical constraints based on these statistics. In (B), it describes the detailed process from entity to entity ($1 \Rightarrow 2$) and from entity to number ($2 \Rightarrow 3$). Based on the distribution information of numerical values, the statistics can be obtained. In (C), the workflow of the Entity Filter is demonstrated. After obtaining the statistics ($mean$), the Filter Function will filter out the entities that meet the numerical constraints based on the predicted entity's attribute values.

sub-query is q_i. By applying a relational projection $\mathcal{P}$, a logical Intersection $\mathcal{I}$, or a logical Union $\mathcal{U}$ (defined below), the embedding of the logical query at step $i+1$ is q_{i+1}. Specifically, at the initial stage, the query embedding q_0 is the embedding of the anchor entity $z_{v_1}, \ldots, z_{v_n}$.

Relational projection operator $\mathcal{P}$: Given a sub-query q_i and a relation between entities $r \in \mathcal{R}$, if the answer set of q_i is $[\![q_i]\!]$, after the projection $q_{i+1} = \mathcal{P}_r(q_i)$, the answer set of q_{i+1} will be represented as $[\![q_{i+1}]\!] = \{v \in \mathcal{V} \mid u \in [\![q_i]\!], r(u, v) = true\}$.

Logical intersection operator $\mathcal{I}$: Given a set of sub-queries $\{q_i^1, \ldots, q_i^n\}$, after the logical intersection operation $q_{i+1} = \mathcal{I}(\{q_i^1, \ldots, q_i^n\})$, the answer set will be represented as $[\![q_{i+1}]\!] = \cap_{j=1,\ldots n}[\![q_i^j]\!]$.

Logical union operator $\mathcal{U}$: Given a set of sub-queries $\{q_i^1, \ldots, q_i^n\}$, after the logical union operation $q_{i+1} = \mathcal{U}(\{q_i^1, \ldots, q_i^n\})$, the answer set will be represented as $[\![q_{i+1}]\!] = \cup_{j=1,\ldots n}[\![q_i^j]\!]$.

In the implementation process, we can use different types of query embedding models to learn $\mathcal{P}$, $\mathcal{I}$, and $\mathcal{U}$, which will be regarded as the backbone of the Number-Entity Predictor.

3.1.2 From entity to number

For mapping from entities to their attribute values in the complex queries, such as $Area(V, C)$, suppose that at step i, the current node is entity, and we represent the current sub-query's embedding as q_i, the entity's attribute as $a \in \mathcal{A}$. Therefore, the attribute mapping can be represented as $q_{i+1} = f_{attr}(q_i, a)$, where q_i and q_{i+1} are input and output of this attribute mapping. Specifically, $f_{attr}(q_i, a)$ is a trainable MLP, which is used to learn information related to attributes.

After obtaining q_{i+1} at step $i+1$, since these numerical attributes have characteristics of both magnitude and distribution, we model each aspect as follows:

Modeling the property of magnitude. We use a deterministic, independent embeddings called DICE (Sundararaman et al., 2020) to effectively capture magnitude properties. This method project the real number in $\mathbb{R}$ to D-dimensional space $\mathbb{R}^D$, and representing the distance between numerical values as cosine similarity between their vectors. Therefore, for any numerical value, we can directly generate its numerical representation without training. To embed number $s_i, i = 1...n,$, we employ a linear mapping to map $s_i \in [0, |a-b|]$ to angles $\theta \in [0, \pi]$:

$$\theta(s_i) = \frac{s_i}{|a-b|}\pi \tag{5}$$

where $[a, b] \subset \mathbb{R}$ is the range of all observed numbers. Then we generate vectors in $\mathbb{R}^D$ by applying a Polar-to-Cartesian transformation in D dimen-

sions:

$$z_i^d = \begin{cases} [\sin(\theta)]^{d-1}\cos(\theta), & 1 \le d < D \\ [\sin(\theta)]^D, & d = D \end{cases} \quad (6)$$

z_i is the embedding of s_i, where d of z_i^d indicates the coordinate in z_i. Therefore, this method can encode any continuous numerical values' magnitude.

Modeling the property of distribution. Since the current sub-query q_{i+1} may be an intersection or union of multiple attribute value sets, we model the distribution of them using a Gaussian Mixture Model (GMM). This model assumes that numbers are sampled from a weighted mixture of K independent Gaussians, and we optimize the marginal log-likelihood objective by summing over the mixtures. We define two MLPs $\pi_u, \pi_\sigma$to predict the mean and variance of this mixture model. Additionally, we use the sigmoid function to constrain the mean within the range [0, 1]. The distribution's generation process (7) and training objective (8) can be represented as follows:

$$\begin{aligned} u &= \pi_u(q_{i+1}, a) \\ \sigma &= \pi_\sigma(q_{i+1}, a) \\ u &= [u_1, u_2, ...u_k],\ \sigma = [\sigma_1, \sigma_2, ...\sigma_k] \\ y' &\sim \mathcal{N}_{[0,1]}(u, \sigma) \end{aligned} \quad (7)$$

$$logP(y|q_{i+1}) = log \sum_{k=1}^{K} [p_k \cdot \frac{C}{\sigma_k} exp(-(\frac{y-u_k}{\sigma_k})^2)] \quad (8)$$

Here, q_{i+1} is the embedding of the query, y is a ground truth of this query, and p_k is the weight of the k-th Gaussian distribution. The larger the training objective $logP$, the closer it is to the true distribution.

3.1.3 Training objective of Numerical-Entity Predictor

Our goal is to learn entity and operators' $(\mathcal{P}, \mathcal{I}, \mathcal{U})$ embeddings as well as the parameters of distribution (π_u, π_σ). Therefore, given a training set of queries and their answers, we optimize the model using the loss function defined in equation (9).

$$L = -\frac{1}{N} \sum_{k=1}^{N} log(\phi(q_k, v_k) + logP \cdot \mathbb{I}_{\mathbb{R}}(v_k)) \quad (9)$$

where (q_k, v_k) is one of N positive query-answer pairs. $\mathbb{I}_{\mathbb{R}}(v_k)$ is an indicator function, which equals to 1 when v_k is number, and 0 when v_k is entity. $logP$ is defined in equation (8). And ϕ can output the the probability of the correct answer among all candidates, using softmax to compute the result of the scoring function $score$.

$$\phi(q_k, v_k) = softmax(score(q_k, v_k)) \quad (10)$$

In (10), the specific form of the scoring function $score$ is determined by the backbone model (Section 3.1).

3.2 Entity Filter

After using the Number-Entity Predictor to predict numerical constraints, the Entity Filter will be used to predict the attributes of all entities in the knowledge graph and filter out the entities that meet the constraints. For example, in Figure 2 (C), in this process, the Entity Filter first uses a pre-trained link predictor to predict the missing attribute values of entities, and then scores the entities according to the Filter function.

3.2.1 Attribute link predictor

Since the prediction of entity's attribute value is a one-hop link prediction, we trained a link predictor $f(v, a)$ using TransEA (Wu and Wang, 2018) to predict attribute values. Specifically, TransEA trains entity triples $\langle h, r, t\rangle$ and attribute triples $\langle v, a, c\rangle$ separately. When given an entity embedding e and an attribute embedding a, the link predictor $f(v, a)$ will output the predicted attribute value $\hat{c}$.

3.2.2 Filter function

The filter function is used to compare the number obtained from the subquery (e.g., $c = \text{mean}(C_?)$) with the predicted attribute value $\hat{c}$ which obtained from $\hat{c} = f(v, a)$, thereby selecting the set of entities that meet the restriction *con*. Inspired by Demir et al. 2023, we use 3 kinds of filter functions : *greater than*, *equal to*, and *less than*, and we define them as follows:

Equal to.

$$con_{equal,a}(\hat{c}, c) := \frac{1}{exp(|\hat{c} - c|/\sigma_a)} \cdot \tau_a(v) \quad (11)$$

where σ_a denotes the standard deviation of $C_a := \{c \in \mathbb{R} | a(v, c) = true, v \in \mathcal{V}\}$, which means all numerical values of attribute a on knowledge graph. And $\tau_a(v)$ represents the probability that entity v has attribute a, which will be introduced later. In equation (11), dividing by the standard deviation σ_a is a method to normalize

the difference $|\hat{c} - c|$, and we constrain the value of $con_{equal,a}(\hat{c}, c)$ within [0,1], representing the likelihood that the entity satisfies this constraint. As the difference between c and $\hat{c}$, denoted as $|\hat{c} - c|$, approaches 0, $con_{equal,a}(\hat{c}, c)$ gets closer to 1.

Less than.

$$con_{less,a}(\hat{c}, c) := \frac{1}{1 + exp((\hat{c} - c)/\sigma_a)} \cdot \tau_a(v) \tag{12}$$

In equation (12), $con_{less,a}(\hat{c}, c)$ represent the likelihood that the entity's attribute value $\hat{c}$ is less than c, so when $(\hat{c} - c)$ approaches $-\infty$, $con_{less,a}(\hat{c}, c)$ approaches 1.

Greater than.

$$con_{greater,a}(\hat{c}, c) := 1 - con_{less,a}(\hat{c}, c) \tag{13}$$

Similar to *less than*, we define the *greater than* constraint using (13)

Among the three filter functions mentioned above, since entity v may not necessarily have attribute a, we use $\tau(e)$ to calculate the probability that entity e has attribute a. To achieve this, we add a virtual node v_{virtual} to the knowledge graph. If entity v has attribute a, we add $a(v, v_{\text{virtual}})$. Consequently, after training the link predictor, predicting the existence of an attribute can be transformed into predicting the presence of a link between two nodes.

4 Experiments

In this section, we will demonstrate CNEQ's effectiveness to answer complex logical queries containing numerical values. Following the settings of Ren et al., 2020, our evaluation primarily focuses on queries in incomplete knowledge graphs. We recommend readers to refer to Appendix A for understanding the evaluation protocol.

4.1 Knowledge graph and query generation

In the experimental section, we use three widely used knowledge graphs: FB15K, DB15K, and YAGO15K. Detailed information about these knowledge graphs will be provided in Appendix B.

To verify our model's prediction capabilities for entities and numerical values, we generated 8 types of query with entity answers and 9 types of query with numerical answers. Also, we create 9 types of constrained queries, each of them have 3 constrains. The detailed structures of these queries are shown in Figure 3. And the specific generation process will be described in Appendix C.

4.2 Baseline models

In the experimental section, we compare our proposed model with two query encoding-based models, GQE and Q2B:

- GQE (Hamilton et al., 2018): GQE embeds a query into a single vector and models different operators as translations and deep sets (Zaheer et al., 2017).
- Q2B (Ren et al., 2020): Q2B encodes queries into hyper-rectangles and represents different operators as interactions between hyper-rectangles.

Since we made some modifications to the baseline models in different experiments, we will provide a detailed introduction to the specific forms of the baselines in the following sections.

4.3 Main results

Performance on Numerical Queries. In numerical queries and the following entity queries, we use GQE and Q2B as baselines, while also using these two models respectively as the backbone for CNEQ's Number-Entity Predictor. In Table 1, we evaluate the performance using mean reciprocal rank (MRR) (Hits@10 in Table 6). The results indicate that CNEQ achieves more accurate results than GQE and Q2B in predicting queries with multiple numerical answers, especially when using Q2B as the backbone, where the relative improvement is more significant. In addition, for more complex query types (e.g., n_2p, n_3p), the relative improvement of CNEQ is greater (average 3.3 and 3.28), indicating that our model has a stronger ability to mitigate the incompletion in multi-hop numerical queries. Overall, CNEQ outperforms the two baseline models across all three datasets.

Performance on Entity Queries. Additionally, we compared 8 types of complex queries with entity answers. We found that after considering the numerical attributes of entities, the model's ability to predict entity answers also improved. The tables in Appendix D show that CNEQ's prediction performance for entity answers also surpasses that of the two baseline models. This indicates that the numerical attributes of entities can enhance the model's predictive ability for entities as additional information.

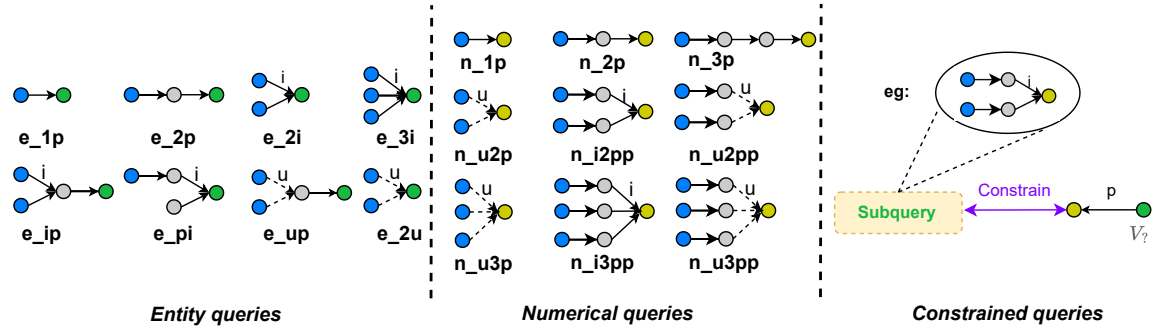

Figure 3: Illustration of 8 types of entity queries, 9 types of numerical queries and 9 types of constrained queries. The "subquery" of constrained queries can be replaced with various types of numerical queries. In each query type, "i" indicates intersection, "u" indicates union, and "p" indicates projection.

Dataset	Backbone	Method	Number									
			n_1p	n_2p	n_3p	n_i2pp	n_i3pp	n_u2p	n_u3p	n_u2pp	n_u3pp	number_all
FB15K	GQE	Baseline	2.68	13.16	3.75	21.27	**26.47**	0.73	0.74	9.15	6.35	9.72
		CNEQ	**2.98**	**17.47**	**9.47**	**21.94**	26.23	**0.83**	**0.80**	**9.79**	**6.73**	**10.95**
	Q2B	Baseline	2.58	15.61	3.06	23.32	29.02	0.89	0.67	13.88	11.25	11.65
		CNEQ	**2.70**	**21.17**	**9.69**	**28.56**	**32.47**	**0.94**	**0.77**	**21.93**	**20.19**	**16.21**
DB15K	GQE	Baseline	1.83	6.46	2.38	10.23	19.63	0.60	0.45	3.13	2.27	5.09
		CNEQ	**1.87**	**9.34**	**3.9**	**13.88**	**21.77**	**0.69**	**0.58**	**3.65**	**2.62**	**6.19**
	Q2B	Baseline	1.62	6.89	2.18	15.27	23.97	0.63	0.53	2.37	3.40	6.21
		CNEQ	**1.76**	**9.08**	**3.92**	**19.9**	**25.74**	**0.70**	**0.56**	**5.34**	**3.46**	**7.56**
YAGO15K	GQE	Baseline	**1.29**	10.15	2.59	21.33	31.28	0.30	0.21	4.47	2.59	8.54
		CNEQ	0.76	**11.83**	**4.50**	**22.89**	**32.88**	**0.35**	**0.25**	**4.95**	**2.87**	**9.11**
	Q2B	Baseline	**0.78**	13.82	2.33	30.72	41.21	0.27	**0.18**	7.27	5.05	11.82
		CNEQ	0.68	**17.02**	**4.81**	**36.67**	**44.99**	**0.32**	0.17	**9.80**	**5.71**	**13.90**

Table 1: MRR results of numerical queries on the FB15K, DB15K, and YAGO15K datasets. "Backbone" of CNEQ refers to the backbone model used in the Number-Entity Predictor. "Baseline" represents the results predicted using this backbone model. "number_all" represents the average performance of all queries.

Performance of Statistical Predictions Section 3.1 introduces that the Number-Entity Predictor can model the distribution of numerical values, thus it can be used to predict the statistics by using distribution's characteristics. Therefore, we predicted the statistics in 9 types of numerical queries. It is worth noting that the model is trained only on entity queries and numberical queries shown in Figure 3, without requiring additional training in this process. For a simple comparison, we employed two baselines: (1) attribute mean: following Demir et al., 2023, we used a model that always predicts the mean value $\frac{1}{|\mathcal{C}_a|}\sum_{c\in\mathcal{C}_a}$ of attribute a. (2) Q2B (top_k): Since the Q2B model can only score and sort all numerical values but cannot determine how many answers it has, so we sampled the top k answers to compute their statistics, where k is a hyperparameter, and we set it to the average number of answers. In Table 2, it can be observed that our CNEQ model outperforms the two baseline models on FB15K dataset.

Performance of Constrained Queries To valid CNEQ's ability to answer complex queries with numerical constraints, we conducted tests on a generated dataset that includes three types of constraints: "equal to", "greater than", and "less than". As a simple comparison, we replaced our Number-Entity Predictor with Q2B(top_k), while keeping the Entity Filter unchanged. The results in Table 3 show that CNEQ achieves better performance across all types of complex queries, indicating that our Number-Entity Predictor has stronger predictive capabilities for numerical constraints compared to Q2B(top_k). Also, with the aid of the Entity Filter, CNEQ is able to effectively answer these queries with constraints. Among these, CNEQ shows the most significant improvement for "equal to" type queries, with an average increase of 3.72 (Hits@10). This is because "equal to" requires a higher predictive capability for numerical values, and CNEQ, with its stronger numerical prediction ability, is better equipped to handle such queries.

Method	n_1p	n_2p	n_3p	n_i2pp	n_i3pp	n_u2p	n_u3p	n_u2pp	n_u3pp	all
	MAE ↓									
Attribution mean	0.3616	0.3520	0.3477	0.3474	0.3464	0.3370	0.3332	0.3457	0.3457	0.3443
Q2B(top_k)	0.0617	0.0468	0.0402	0.0567	0.0594	0.0535	0.0481	0.0241	0.0170	0.0434
CNEQ	**0.0334**	**0.0156**	**0.0173**	**0.0183**	**0.0166**	**0.0219**	**0.0176**	**0.0150**	**0.0126**	**0.0173**
	MSE ↓									
Attribution mean	0.1528	0.1469	0.1427	0.1428	0.1423	0.1388	0.1364	0.1415	0.141	0.1415
Q2B(top_k)	0.0099	0.0056	0.0044	0.0076	0.0079	0.0056	0.0041	0.0022	0.0011	0.0049
CNEQ	**0.0049**	**0.0010**	**0.0015**	**0.0012**	**0.0011**	**0.0018**	**0.0012**	**0.0007**	**0.0006**	**0.0012**

Table 2: MRR results of predicting the mean of numerical answers.

Constrain	Model	n_1p	n_2p	n_3p	n_i2pp	n_i3pp	n_u2p	n_u3p	n_u2pp	n_u3pp
Equal to	Q2B(top_k)+ Entity filter	11.48	10.84	10.70	11.45	11.87	11.67	11.45	12.22	12.50
	CNEQ	**15.94**	**16.29**	**14.09**	**16.50**	**16.17**	**14.84**	**14.28**	**15.10**	**14.43**
Greater than	Q2B(top_k)+ Entity filter	9.57	10.78	11.48	11.05	10.40	11.42	12.05	11.04	12.26
	CNEQ	**10.34**	**11.86**	**12.09**	**11.54**	**11.84**	**12.02**	**12.73**	**11.85**	**12.43**
Less than	Q2B(top_k)+ Entity filter	7.03	7.68	8.72	8.24	8.24	8.63	9.48	9.15	9.51
	CNEQ	**7.93**	**8.70**	**9.32**	**8.86**	**9.34**	**9.22**	**11.50**	**9.39**	**9.78**

Table 3: Hits@10 results of answering constrained queries.

Method	n_1p	n_2p	n_3p	n_i2pp	n_i3pp	n_u2p	n_u3p	n_u2pp	n_u3pp
CNEQ(w/o both)	2.58	15.61	3.07	23.32	29.02	0.89	0.67	13.88	11.25
CNEQ(w/o magnitude)	2.55	19.88	8.99	27.78	31.39	0.86	0.62	20.51	19.43
CNEQ(w/o distribution)	2.68	19.75	9.11	26.98	31.78	0.82	**0.88**	20.48	19.08
CNEQ	**2.70**	**21.17**	**9.69**	**28.56**	**32.47**	**0.94**	0.77	**21.93**	**20.19**

Table 4: MRR results of the ablation study. To demonstrate the importance of value's characteristics, We removed the "distribution" and "magnitude" modules of the numerical values.

4.4 Ablation study

Table 4 presents the results of the ablation study. "magnitude" and "distribution" represent the two modules mentioned in Section 3.1.2. It can be observed that the prediction performance decreases after removing either the distribution or magnitude components. Particularly, when both components are removed, there is a significant drop in performance, indicating that our modeling of numerical features greatly enhances the prediction performance of numerical attributes.

5 Related Work

Our work builds upon a wealth of previous research in knowledge graph reasoning, numerical representation, etc. The methods most relevant to our work are about complex reasoning over knowledge graphs. These methods often use different structures to encode complex queries. For example, Hamilton et al., 2018 encode queries into a vector, and Ren et al., 2020 encode queries into hyper-rectangles. In addition, Ren and Leskovec, 2020, Yang et al., 2022, Long et al., 2022, and Choudhary et al., 2021 use different types of distributions to represent complex queries. Xu et al., 2023 encode diverse complex queries into a unified triple, and Bai et al., 2022 encode each query into multiple vectors, named particle embeddings. These different modeling approaches are used to solve different problems when answering queries.

Second line of related work is learning numerical values. Due to the importance of numerical values, many works have focused on improving model's ability to learn numerical values (Spokoyny and Berg-Kirkpatrick, 2020; Geva et al., 2020; Zhang et al., 2020; Huang et al., 2024b), thereby enhancing the model's performance on downstream tasks. It is same in Knowledge Graph, García-Durán and Niepert, 2017,Chung et al., 2023, Tay et al., 2017 have learned the attributes of entities. Lin et al., 2016 propose a knowledge representation model which can learn entity, relation, and attribute simultaneously. Kotnis and García-Durán, 2018 predicting numerical values by leveraging global and local information to fill in missing numerical attributes. And Bai et al., 2023 addressed numerical comparison by adding edges between attribute values. These works all attempt to utilize numerical

information on knowledge graphs.

6 Conclusion

We propose CNEQ, designed to address complex queries involving numerical values. Additionally, CNEQ effectively handles complex queries related to statistics and queries with numerical constraints. We also construct benchmarks for various types of complex queries, which facilitates further research on complex queries over knowledge graphs. Our results demonstrate that CNEQ outperformes several query embedding models, and achieves SOTA performance.

7 Limitations

This paper has some limitations. For the relationships between numerical values, in addition to greater than, less than, and equal to, there should also be multiplication and division relationships. However, we have not designed experiments or validations for these relationships. We will complete the relevant validations in our future work.

8 Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62276110, No. 62172039 and in part by the fund of Joint Laboratory of HUST and Pingan Property Casualty Research (HPL). The authors would also like to thank the anonymous reviewers for their comments on improving the quality of this paper.

References

Jiaxin Bai, Chen Luo, Zheng Li, Qingyu Yin, Bing Yin, and Yangqiu Song. 2023. Knowledge graph reasoning over entities and numerical values. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 57–68.

Jiaxin Bai, Zihao Wang, Hongming Zhang, and Yangqiu Song. 2022. Query2particles: Knowledge graph reasoning with particle embeddings. *arXiv preprint arXiv:2204.12847*.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia-a crystallization point for the web of data. *Journal of web semantics*, 7(3):154–165.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan Reddy. 2021. Probabilistic entity representation model for reasoning over knowledge graphs. *Advances in neural information processing systems*, 34:23440–23451.

Chanyoung Chung, Jaejun Lee, and Joyce Jiyoung Whang. 2023. Representation learning on hyper-relational and numeric knowledge graphs with transformers. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 310–322.

Caglar Demir, Michel Wiebesiek, Renzhong Lu, Axel-Cyrille Ngonga Ngomo, and Stefan Heindorf. 2023. Litcqd: Multi-hop reasoning in incomplete knowledge graphs with numeric literals. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 617–633. Springer.

Alberto García-Durán and Mathias Niepert. 2017. Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. *arXiv preprint arXiv:1709.04676*.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. *arXiv preprint arXiv:2004.04487*.

Jia Guo and Stanley Kok. 2021. Bique: Biquaternionic embeddings of knowledge graphs. *arXiv preprint arXiv:2109.14401*.

Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31.

Rikui Huang, Wei Wei, Xiaoye Qu, Wenfeng Xie, Xianling Mao, and Dangyang Chen. 2024a. Joint multi-facts reasoning network for complex temporal question answering over knowledge graph. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 10331–10335. IEEE.

Rikui Huang, Wei Wei, Xiaoye Qu, Shengzhe Zhang, Dangyang Chen, and Yu Cheng. 2024b. Confidence is not timeless: Modeling temporal validity for rule-based temporal knowledge graph forecasting. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10783–10794.

Bhushan Kotnis and Alberto García-Durán. 2018. Learning numerical attributes in knowledge bases. In *Automated Knowledge Base Construction (AKBC)*.

Agustinus Kristiadi, Mohammad Asif Khan, Denis Lukovnikov, Jens Lehmann, and Asja Fischer. 2019. Incorporating literals into knowledge graph embeddings. In *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New*

Zealand, October 26–30, 2019, Proceedings, Part I 18, pages 347–363. Springer.

Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2016. Knowledge representation learning with entities, attributes and relations. *ethnicity*, 1:41–52.

Xiao Long, Liansheng Zhuang, Li Aodi, Shafei Wang, and Houqiang Li. 2022. Neural-based mixture probabilistic query embedding for answering fol queries on knowledge graphs. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3001–3013.

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*.

Hongyu Ren and Jure Leskovec. 2020. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726.

Georgios P Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. *arXiv preprint arXiv:1805.08154*.

Daniel Spokoyny and Taylor Berg-Kirkpatrick. 2020. An empirical investigation of contextualized number prediction. *arXiv preprint arXiv:2011.07961*.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for numeracy-preserving word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4742–4753.

Yi Tay, Luu Anh Tuan, Minh C Phan, and Siu Cheung Hui. 2017. Multi-task neural network for non-discrete attribute prediction in knowledge graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1029–1038.

Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. 2021. Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*.

Siyuan Wang, Zhongyu Wei, Meng Han, Zhihao Fan, Haijun Shan, Qi Zhang, and Xuanjing Huang. 2023. Query structure modeling for inductive logical reasoning over knowledge graphs. *arXiv preprint arXiv:2305.13585*.

Yanrong Wu and Zhichun Wang. 2018. Knowledge graph embedding with numeric attributes of entities. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 132–136.

Yao Xu, Shizhu He, Cunguang Wang, Li Cai, Kang Liu, and Jun Zhao. 2023. Query2triple: Unified query encoding for answering diverse complex queries over knowledge graphs. *arXiv preprint arXiv:2310.11246*.

Dong Yang, Peijun Qing, Yang Li, Haonan Lu, and Xiaodong Lin. 2022. Gammae: Gamma embeddings for logical queries on knowledge graphs. *arXiv preprint arXiv:2210.15578*.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems*, 30.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. 2020. Do language embeddings capture scales? *arXiv preprint arXiv:2010.05345*.

A Evaluation Protocol

For any query q, assume that its answer sets obtained using subgraph matching on $\mathcal{G}_{train}$, $\mathcal{G}_{valid}$, and $\mathcal{G}_{test}$ are $[\![q]\!]_{train}$, $[\![q]\!]_{valid}$, and $[\![q]\!]_{test}$, respectively. We train using q with its answers $[\![q]\!]_{train}$, and during evaluation, we test only on $[\![q]\!]_{valid} \setminus [\![q]\!]_{train}$ ($[\![q]\!]_{test} \setminus [\![q]\!]_{valid}$), which means the answers found in $\mathcal{G}_{valid}$ but not in $\mathcal{G}_{train}$ (similarly for $[\![q]\!]_{test} \setminus [\![q]\!]_{valid}$). Therefore, we focus solely on non-trivial answers that require inferring at least one edge to find the answer. Given a test query q, for each of its non-trivial answers v, we rank v among all non-answer sets $\mathcal{V} \setminus [\![q]\!]_{test}$ as r.

The evaluation metrics can be Mean Reciprocal Rank (MRR): $\frac{1}{r}$ or Hits at K (H@K): $\mathbf{1}[\mathrm{r} \leq \mathrm{K}]$ for predicting entities or numerical values. And when predicting the statistics, we use Mean Absolute Error (MAE) and Mean Square Error (MSE) to evaluate.

B Statistics of Knowledge Graphs

Dataset	Split	Ent_nodes	Num_nodes	Rel edges	Att edges
YAGO	train	15351	17596	196616	21847
	valid	15386	18270	221194	22807
	test	15404	18770	245772	23520
DB15K	train	12766	20540	145120	33129
	valid	12811	22876	161868	37270
	test	12842	25159	178394	41411
FB15K	train	14940	12310	947540	20248
	valid	14949	13518	1065982	22779
	test	14951	14689	1184426	25311

Table 5: The information of the three knowledge graphs which we used to generate queries

In Table 5, we present detailed data on the three knowledge graphs: FB15K (Bordes et al., 2013), DB15K (Bizer et al., 2009), and YAGO15K (Suchanek et al., 2007). For a given knowledge graph $\mathcal{G}$, we first split its edges into *training edges*, *validation edges*, and *test edges* in a ratio of 8:1:1. Then, we partition these three types of edges to form the training graph $\mathcal{G}_{train}$, validation graph $\mathcal{G}_{valid}$, and test graph $\mathcal{G}_{test}$, respectively. Their edges consist of: *training edges*, *training edges* + *validation edges*, and *training edges* + *validation edges* + *test edges*.

Among them, **Ent_nodes** represents the number of entity nodes, **Num_nodes** represents the number of numerical nodes, **Rel_edges** represents the triples between entities, and **Att_edges** represents the triples between entities and numerical values.

Algorithm 1: Query generate algorithm

Input:
  $\mathcal{G}$: Knowledge Graph
  $\mathcal{T}$: Template of the query
Output:
  q: query has been generated
  ans: answers of query q
$t \leftarrow answer\_type(\mathcal{T})$
if *t is entity* **then**
  $v_{root} \leftarrow sample(\{v|v \in \mathcal{V}\})$
  $q \leftarrow sample\_query(v_{root}, \mathcal{T}, \mathcal{G})$
  $ans \leftarrow find\_answers(q, \mathcal{G})$
end
else if *t is number* **then**
  $v_{root} \leftarrow sample(\{c|c \in \mathcal{C}\})$
  $q \leftarrow sample\_query(v_{root}, \mathcal{T}, \mathcal{G})$
  $ans \leftarrow find\_answers(q, \mathcal{G})$
end
return q, *ans*

C Query generation

In this section, we will introduce the process of generating different queries. For example, in Algorithm 1, given a knowledge graph $\mathcal{G}$ and a query template $\mathcal{T}$, the model first determines the answer's type based on the template. For example, if the answer's type is entity, it samples an entity v from the entity set $\mathcal{V}$ as the root node of the DAG, which is the target node of the query. The function $sample\_query$ starts from the root node v and iteratively gets the previous operator according to the DAG's topological structure. If the operator is $\mathcal{P}$, it instantiates $\mathcal{P}$ as p and samples a node v' that satisfies $p(v, v')$; if the operator is $\mathcal{I}$ or $\mathcal{U}$, it instantiates multiple p and samples nodes v' that satisfies $p(v, v')$ separately. This process is recursively executed until reach leaf node, thereby generating the query q. After that, the function $find\_answers$ can start from the leaf nodes (anchor entities) in q and search for answers on the knowledge graph, finally obtaining the answer set ans.

For queries that answer is statistic, it is only necessary to calculate the statistics of all answer numbers in ans for the query q. Additionally, for queries that include numerical constraints, after obtaining the number c of the constrain, we find the entity set $\{v|a(v, c') = true, con(c, c') = true, v \in \mathcal{V}\}$ on the knowledge graph whose attribute value c' satisfy $con(c, c') = true$, which

serves as the answer to the query. Specifically, the answer entities in the test set only exist in $\mathcal{G}_{train}$ but not in $\mathcal{G}_{test}$, as discussed in Section A.

Table 9 illustrates all types of complex queries we generated. For "Entity queries" and "Number queries," we sampled across three knowledge graphs. For any given knowledge graph $\mathcal{G}$, we generated training queries, validation queries, and test queries on $\mathcal{G}_{train}$, $\mathcal{G}_{valid}$, and $\mathcal{G}_{test}$, respectively. Additionally, for "Constrained queries," We generated queries for the three types of constraints: "equal to," "greater than," and "less than". since no training is required, we only generated test queries.

D More information about performance of entity queries

In Tables 7 and 8, we present the MRR and Hits@10 results of entity queries on the FB15K, DB15K, and YAGO15K. It can be observed that for entity queries, CNEQ, whether using GQE or Q2B as the backbone, outperforms the baseline. This may be because CNEQ takes into account the attribute information of entities, enriching the representation of entity nodes both in terms of spatial structure and semantic information, thereby enhancing the prediction performance of entity. Particularly, when Q2B is used as the backbone, the overall performance of improves more, with increases of 1.97%, 2.85%, and 4.82% (MRR) on the three datasets, respectively. This is because Q2B has a stronger learning capability for entity's features compared to GQE, making the promotion of numerical attributes more evident.

Dataset	Backbone	Method	Number									
			n_1p	n_2p	n_3p	n_i2pp	n_i3pp	n_u2p	n_u3p	n_u2pp	n_u3pp	number_all
FB15K	GQE	Baseline	5.04	24.09	7.97	34.14	**42.45**	1.38	1.48	17.55	12.73	16.83
		CNEQ	**5.54**	**28.55**	**16.37**	**34.82**	42.21	**1.63**	**1.63**	**18.10**	**13.18**	**18.45**
	Q2B	Baseline	4.74	26.66	6.31	37.31	45.51	1.66	1.30	23.66	20.17	19.36
		CNEQ	**5.04**	**31.64**	**17.16**	**41.27**	**47.81**	**1.71**	**1.49**	**29.20**	**26.85**	**23.51**
DB15K	GQE	Baseline	2.74	12.49	4.70	20.25	35.33	1.02	0.92	6.15	4.59	9.58
		CNEQ	**2.96**	**16.33**	**7.48**	**25.33**	**37.67**	**1.14**	**0.98**	**6.76**	**5.06**	**11.14**
	Q2B	Baseline	2.87	13.33	4.43	27.47	40.15	1.12	**0.99**	6.74	4.69	11.05
		CNEQ	**2.93**	**16.31**	**7.60**	**32.60**	**41.97**	**1.23**	0.74	**9.54**	**5.43**	**12.79**
YAGO15K	GQE	Baseline	**2.01**	20.1	5.86	36.75	**52.36**	**0.48**	0.32	9.32	5.29	15.13
		CNEQ	1.78	**23.83**	**8.97**	**38.89**	51.39	**0.48**	**0.35**	**10.44**	**5.85**	**16.01**
	Q2B	Baseline	**1.92**	26.16	5.14	48.29	60.03	**0.53**	**0.30**	15.51	10.39	19.22
		CNEQ	1.33	**32.09**	**9.88**	**52.62**	**62.78**	0.43	0.29	**17.49**	**10.46**	**21.18**

Table 6: Hits@10 results of numerical queries on the FB15K, DB15K, and YAGO15K datasets

Dataset	Backbone	Method	Entity								
			e_1p	e_2p	e_2i	e_3i	e_ip	e_pi	e_2u	e_up	entity_all
FB15K	GQE	Baseline	30.91	9.81	38.04	43.03	15.97	26.52	8.76	**10.98**	22.35
		CNEQ	**30.99**	**11.10**	**38.63**	**43.47**	**16.07**	**27.32**	**11.48**	9.45	**22.88**
	Q2B	Baseline	**38.36**	8.69	37.00	41.50	13.64	**27.08**	17.27	8.10	22.88
		CNEQ	38.29	**9.94**	**37.09**	**42.57**	**14.35**	27.00	**17.32**	**8.40**	**23.33**
DB15K	GQE	Baseline	11.95	3.14	26.17	**41.23**	4.45	11.47	3.19	2.79	11.94
		CNEQ	**12.63**	**3.20**	**26.41**	41.15	**4.75**	**11.61**	**3.29**	**3.05**	**11.96**
	Q2B	Baseline	13.82	2.76	27.61	41.21	4.43	12.70	3.42	2.86	12.27
		CNEQ	**14.10**	**2.98**	**29.25**	**42.52**	**4.53**	**12.85**	**3.57**	**2.93**	**12.62**
YAGO15K	GQE	Baseline	12.34	2.58	31.88	39.96	3.71	12.48	3.93	**2.40**	13.24
		CNEQ	**12.89**	**2.84**	**32.64**	**41.41**	**3.97**	**12.75**	**4.42**	2.37	**13.41**
	Q2B	Baseline	19.40	2.01	39.67	48.56	4.10	14.12	6.96	2.17	16.19
		CNEQ	**22.95**	**2.65**	**40.83**	**50.10**	**4.53**	**15.10**	**7.21**	**2.47**	**16.97**

Table 7: MRR results of entity queries on the FB15K, DB15K, and YAGO15K datasets

Dataset	Backbone	Method	Entity								
			e_1p	e_2p	e_2i	e_3i	e_ip	e_pi	e_2u	e_up	entity_all
FB15K	GQE	Baseline	55.76	**20.30**	62.63	70.15	26.95	45.72	23.00	17.08	38.96
		CNEQ	**56.41**	**20.30**	**63.54**	**70.48**	**27.56**	**46.28**	**24.09**	**17.88**	**39.58**
	Q2B	Baseline	64.41	17.61	63.12	69.65	24.47	47.53	33.79	15.73	40.37
		CNEQ	**65.26**	**18.8**	**63.78**	**70.46**	**24.63**	**47.68**	**33.84**	**15.83**	**40.84**
DB15K	GQE	Baseline	24.63	6.31	49.71	68.29	9.08	22.88	7.03	5.45	21.84
		CNEQ	**26.54**	**6.57**	**50.24**	**68.62**	**9.72**	**23.11**	**7.13**	**6.06**	**22.18**
	Q2B	Baseline	29.26	5.80	52.41	69.77	8.60	25.06	7.87	5.71	22.89
		CNEQ	**30.56**	**6.06**	**52.70**	**70.06**	**8.68**	**25.20**	**8.23**	**5.74**	**22.95**
YAGO15K	GQE	Baseline	26.27	5.93	51.37	61.44	7.36	23.76	9.07	4.57	22.66
		CNEQ	**27.85**	**6.38**	**52.71**	**62.26**	**8.05**	**24.11**	**10.32**	**4.94**	**23.16**
	Q2B	Baseline	36.24	4.36	58.41	67.33	8.54	27.65	17.53	4.44	26.29
		CNEQ	**40.39**	**5.82**	**59.39**	**67.90**	**9.20**	**28.65**	**17.55**	**5.19**	**27.17**

Table 8: Hits@10 results of entity queries on the FB15K, DB15K, and YAGO15K datasets

Entity queries									
KG	**Splid**	**e_1p**	**e_2p**	**e_2i**	**e_3i**	**e_ip**	**e_pi**	**e_2u**	**e_up**
YAGO15K	train	16135	48284	65810	87240	87769	83246	80350	89367
	valid	2292	8468	7233	9486	9907	9737	9473	9918
	test	2348	8437	7274	9498	9896	9736	9447	9912
DB15K	train	25053	60102	71806	87282	88412	85824	82157	89337
	valid	2505	8570	5930	7756	9898	9656	9366	9910
	test	2435	8524	6003	8077	9907	9687	9359	9913
FB15K	train	38373	78663	81348	88354	88611	88353	87621	88739
	valid	4828	9524	9047	9738	9798	9788	9778	9798
	test	4845	9506	9013	9765	9793	9791	9766	9797
Number queries									
KG	**Splid**	**n_1p**	**n_2p**	**n_3p**	**n_i2pp**	**n_i3pp**	**n_u2p**	**n_u3p**	**n_u2pp**	**n_u3pp**
YAGO15K	train	19586	22130	59521	70071	87918	89926	90000	80268	89476
	valid	894	2849	8834	7578	9619	9921	9936	9438	9920
	test	679	2803	8813	7605	9597	9925	9936	9466	9920
DB15K	train	23008	37294	78431	77418	88352	89785	90000	82180	89230
	valid	2759	3697	9361	7016	8810	9923	9936	9454	9908
	test	2701	3755	9411	7145	8932	9928	9936	9411	9893
FB15K	train	16815	38490	80923	80830	88355	88524	88750	85209	88712
	valid	2037	4880	9565	9008	9759	9790	9798	9701	9798
	test	2004	4931	9578	9105	9754	9785	9798	9699	9797
Constrained queries									
KG	**Constrain**	**n_1p**	**n_2p**	**n_3p**	**n_i2pp**	**n_i3pp**	**n_u2p**	**n_u3p**	**n_u2pp**	**n_u3pp**
FB15K	Equal to	4518	17227	45982	41407	49151	49813	49968	48486	49956
	Greater than	4504	17280	45985	41214	49123	49816	49968	48441	49959
	Less than	4535	17221	46040	41419	49148	49801	49968	48381	49956

Table 9: The statistics of 3 types of queries sampled from FB15k, DB15k, and YAGO15k.