Inductive Logical Query Answering in Knowledge Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Formulating and answering logical queries is a standard communication interface for knowledge graphs (KGs) and their representations. Alleviating the notorious incompleteness of real-world KGs, neural methods achieved impressive results in link prediction and complex query answering tasks by learning representations of entities, relations, and queries. Still, most existing query answering methods are inherently transductive and cannot be generalized to KGs containing new entities without retraining entity embeddings. In this work, we study the inductive query answering task where inference is performed on a graph containing new entities with queries over both seen and unseen entities. To this end, we devise two mechanisms leveraging inductive *node* and *relational structure* representations powered by graph neural networks (GNNs). Experimentally, we show that inductive models are able to perform logical reasoning at inference time over unseen nodes generalizing to graphs up to 500% larger than training ones. Exploring the efficiency–effectiveness trade-off, we find the inductive *relational structure* method generally achieves higher performance, while the inductive *node representation* method is able to answer complex queries in the *inference-only* regime without any training on queries and scale to graphs of millions of nodes.

1 Introduction

Traditionally, querying knowledge graphs (KGs) is performed via databases using structured query languages like SPARQL. Databases can answer complex queries relatively fast under the assumption of *completeness*, i.e., there is no missing information in the graph. In practice, however, KGs are notoriously incomplete [31]. Embedding-based methods that learn vector representations of entities and relations are known to be effective in *simple link prediction* predicting heads or tails of query patterns *(head, relation, ?)*, e.g., *(Einstein, graduate, ?)*, forming the field of *KG completion* [1, 16].

Complex queries are graph patterns expressed in a subset of first-order logic (FOL) with operators such as intersection ($\land$), union ($\lor$), negation ($\neg$) and existentially quantified ($\exists$) variables[1], e.g., $?U.\exists V : \texttt{Win}(\texttt{NobelPrize}, V) \land \texttt{Citizen}(\texttt{USA}, V) \land \texttt{Graduate}(V, U)$ (Fig. 1). Complex queries define a superset of link prediction on KGs. The conventional link prediction task can be viewed as a complex query with a single triplet pattern without logic operators, e.g., $\texttt{Citizen}(\texttt{USA}, V)$, which we also denote as a *projection* query.

To tackle complex queries on incomplete knowledge graphs, *query embedding* methods are proposed to execute logic operations in the latent space, including variants that employ geometric [14, 22, 37], probabilistic [23, 8], neural-symbolic [25, 7, 5], neural [20, 4], and GNN [10, 3] approaches for learning entity, relation, and query representations.

[1] The universal quantifier ($\forall$) is often discarded as in real-world KGs there is no node connected to all others.

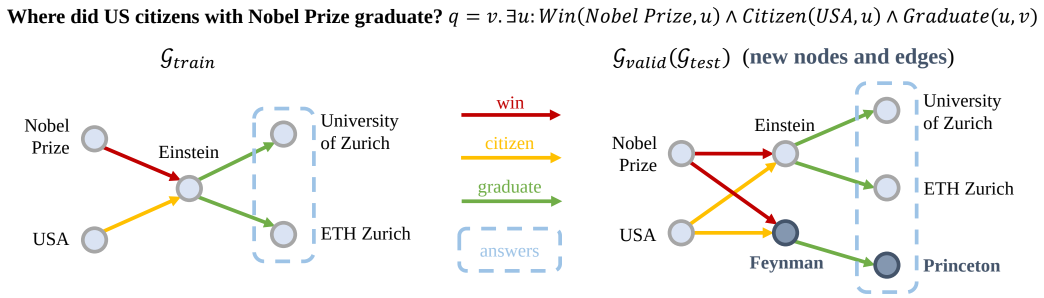
Where did US citizens with Nobel Prize graduate? $q = v.\exists u: Win(Nobel\ Prize, u) \wedge Citizen(USA, u) \wedge Graduate(u, v)$
$\mathcal{G}_{train}$
Nobel Prize
Einstein
USA
University of Zurich
ETH Zurich
win
citizen
graduate
answers
$\mathcal{G}_{valid}(\mathcal{G}_{test})$ (new nodes and edges)
Nobel Prize
Einstein
USA
Feynman
University of Zurich
ETH Zurich
Princeton

Figure 1: Inductive query answering problem: at inference time, the graph is updated with new nodes `Feynman` and `Princeton` and edges such that the same query now has more answers.

However, this very fact of learning a separate embedding for each entity makes those methods inherently *transductive* i.e., they are bound to the space of learned entities and can not be generalized to unseen entities without retraining the whole embedding matrix which can be prohibitively expensive in large graphs. The problem is illustrated in Fig. 1: given a graph about `Einstein` and a logical query *Where did US citizens with Nobel Prize graduate?*, transductive QE methods learn to execute logical operators and return the answer set {`University of Zurich, ETH Zurich`}. Then, the graph is updated with new nodes and edges about `Feynman` and `Princeton`, and the same query now has more correct answers {`University of Zurich, ETH Zurich, Princeton`} as new unseen entities satisfy the query as well.

Such *inductive inference* is not possible for transductive models as they do not have representations for new `Feynman` and `Princeton` nodes. In the extreme case, inference graphs might be disconnected from the training one and only share the set of relations. Therefore, inductive capabilities are a key factor to enable transferring trained query answering models onto updated or entirely new KGs.

In this work, we study answering complex queries in the inductive setting, where the model has to deal with unseen entities at inference time. Inspired by recent advancement in inductive link prediction on KGs [39, 12], we devise two solutions for learning inductive representations for complex query: 1) The first solution, NodePiece-QE, extends the inductive node representation model NodePiece [12] for complex query answering. NodePiece-QE learns **inductive representations of each entity** as a function of tokens from a fixed-size vocabulary, and answers complex query with a non-parametric logical query executor [5]. The advantages of NodePiece-QE are that it only needs to be trained on simple link prediction data, answers complex queries in the *inference-only* mode, and that it can scale to large KGs. 2) The second solution, NBFNet-QE, extends the inductive link prediction model NBFNet [39] for complex query answering. NBFNet-QE learns **inductive representations of the relational structure** without entity embeddings, and uses the relational structure between the query constants and the answers to make the prediction. NBFNet-QE can be trained end-to-end on complex queries, achieves much better performance than NodePiece-QE, but struggles to scale to large KGs.

To the best of our knowledge, this is the first work to study complex logical query answering in the inductive setting. Conducting experiments on a novel benchmarking suite of 10 datasets, we find that (i) both inductive solutions exhibit non-trivial performance answering logical queries over unseen entities and query patterns; (ii) inductive models demonstrate out-of-distribution generalization capabilities to graphs up to 500% larger than training ones; (iii) akin to updatable databases, inductive methods can successfully find new correct answers to known training queries after adding new nodes and edges; (iv) the inductive *node representation* method scales to answering logical queries over a graph of 2M nodes with 500k new, unseen nodes; (v) GNN-based models still exhibit difficulties [19, 34] generalizing to larger graphs than they were originally trained on.

2 Related Work

Knowledge Graph Completion. Knowledge graph completion, a.k.a. simple link prediction, has been widely studied in the *transductive* paradigm [6, 32, 26, 36], i.e., when inference is performed on the same training graph with a fixed set of entities. Generally, these methods learn a shallow embedding vector for each entity. We refer the audience to respective surveys [1, 16] covering dozens of transductive embedding methods. The emergence of message passing [13] and Graph Neural Networks (GNNs) has led to more advanced, *inductive* representation learning approaches that model entity or triplet representations as a function of the graph structure in its neighborhood. GraIL [27] learns triplet representations based on the subgraph structure surrounding the two entities. NeuralLP [33], DRUM [24] and NBFNet [39] learn the pairwise entity representations based on the set of relation paths between two entities. NodePiece [12] learns entity representations from a fixed-size vocabulary of tokens that can be anchor nodes in a graph or relation types.

Complex Query Answering. In the complex (multi-hop) query answering setup with logical operators, existing models employ different approaches, e.g., geometric [14, 22, 37], probabilistic [23, 8], neural-symbolic [25, 7, 5], neural [20, 4], and GNN [10, 3]. Still, all the approaches are created and evaluated exclusively in the transductive mode where the set of entities does not change at inference time. To the best of our knowledge, there is no related work in inductive logical query answering when an inference graph contains new entities. With our work, we aim to bridge this gap and extend inductive representation learning algorithms to logical query answering. In particular, we focus on the inductive setup where an inference graph is a superset of a training graph[2] such that (i) inference queries require reasoning over both seen and new entities; (ii) original training queries might have more correct answers at inference time with the addition of new entities.

3 Preliminaries and Problem Definition

Knowledge Graph and Inductive Setup. Given a finite set of entities $\mathcal{E}$, a finite set of relations $\mathcal{R}$, and a set of triples (edges) $\mathcal{T} = (\mathcal{E} \times \mathcal{R} \times \mathcal{E})$, a knowledge graph $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$. Accounting for the inductive setup, we define a *training* graph $\mathcal{G}_{train} = (\mathcal{E}_{train}, \mathcal{R}, \mathcal{T}_{train})$ and *inference* graph $\mathcal{G}_{inf} = (\mathcal{E}_{inf}, \mathcal{R}, \mathcal{T}_{inf})$ such that $\mathcal{E}_{train} \subset \mathcal{E}_{inf}$ and $\mathcal{T}_{train} \subset \mathcal{T}_{inf}$. That is, the *inference* graph extends the training graph with new entities and edges[3].The inference graph $\mathcal{G}_{inf}$ is an incomplete part of the not observable complete graph $\hat{\mathcal{G}}_{inf} = (\mathcal{E}_{inf}, \mathcal{R}, \hat{\mathcal{T}}_{inf})$ with $\hat{\mathcal{T}}_{inf} = \mathcal{T}_{inf} \cup \mathcal{T}_{pred}$ whose missing triples $\mathcal{T}_{pred}$ have to be predicted at inference time.

First-Order Logic Queries. Applied to KGs, a first-order logic (FOL) query $\mathcal{Q}$ is a formula that consists of constants $\mathcal{C}$ ($\mathcal{C} \subseteq \mathcal{E}$), variables $\mathcal{V}$ ($\mathcal{V} \subseteq \mathcal{E}$, existentially quantified), relation *projections* $R(a, b)$ denoting a binary function over constants or variables, and logic symbols ($\exists, \wedge, \vee, \neg$). The answers $A_{\mathcal{G}}(\mathcal{Q})$ to the query $\mathcal{Q}$ are assignments of variables in a formula such that the instantiated query formula is a subgraph of the complete graph $\hat{\mathcal{G}}$.

Fig. 1 illustrates the logical form of a query *Where did US citizens with Nobel Prize graduate?* as $?U.\exists V : \texttt{Win}(\texttt{NobelPrize}, V) \wedge \texttt{Citizen}(\texttt{USA}, V) \wedge \texttt{Graduate}(V, U)$ where NobelPrize and USA are *constants*; Win, Citizen, Graduate are *relation projections* (labeled edges); V, U - *variables* such that V is an existentially quantified free variable and U is the projected bound *target* of the query. Common for the literature, we aim at predicting assignments of the query *target* whereas assignments of intermediate variables might not always be explicitly interpreted depending on the model architecture. In the example, the answer set $A_{\mathcal{G}}(\mathcal{Q})$ is a binding of a target variable U to constants University of Zurich and ETH Zurich.

Inductive FOL Queries. In the standard transductive query answering setup, query constants and variables at both training and inference time belong to the same set of entities, i.e., $\mathcal{C}_{train} = \mathcal{C}_{inf} \subseteq \mathcal{E}, \mathcal{V}_{train} = \mathcal{V}_{inf} \subseteq \mathcal{E}$. In the inductive setup covered in this work, query constants and variables at inference time belong to a different and larger set of entities $\mathcal{E}_{inf}$ from the inference graph $\mathcal{G}_{inf}$, i.e., $\mathcal{C}_{train} \subseteq \mathcal{E}_{train}, \mathcal{V}_{train} \subseteq \mathcal{E}_{train}$ but $\mathcal{C}_{inf} \subseteq \mathcal{E}_{inf}, \mathcal{V}_{inf} \subseteq \mathcal{E}_{inf}$. This also leads to the fact that training queries executed over the inference graph might have more correct answers, i.e., $A_{\mathcal{G}_{train}}(\mathcal{Q}) \subseteq A_{\mathcal{G}_{inf}}(\mathcal{Q})$. For example (cf. Fig. 1), the inference graph is updated with new nodes Feynman, Princeton and their new respective edges. The same query now has a larger set of intermediate variables satisfying the formula (Feynman) and an additional correct answer Princeton. Therefore, inductive generalization is essential for obtaining representations of such new nodes and enabling logical reasoning over both seen and new nodes, i.e., finding more answers to known queries in larger graphs or answering new queries with new constants. In the following section, we describe two approaches for achieving inductive generalization with different parameterization strategies.

[2] The set of relation types is fixed.

[3] Note that the set of relation types $\mathcal{R}$ remains the same.

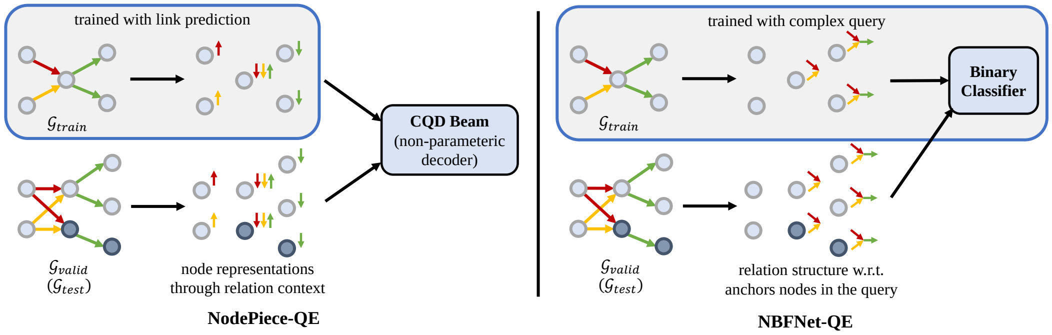

Figure 2: Inductive node representation (NodePiece-QE, left) and relational structure (NBFNet-QE, right) strategies for complex logical query answering. In NodePiece-QE, we obtain inductive node representations through the invariant set of tokens (here, through incident relation types). NodePiece-QE is the inference-only approach and is pre-trained with simple *1p* link prediction and can be directly applied to inductive complex queries with a non-parametric decoder (e.g., CQD Beam). In NBFNet-QE, we learn the the relative structure of each node w.r.t. the anchor nodes in the query. NBFNet-QE is trainable end-to-end with *complex queries*.

4 Method

Inductive Representations of Complex Queries. Given a complex query $\mathcal{Q} = (\mathcal{C}, \mathcal{R}_\mathcal{Q}, \mathcal{G})$), the goal is to rank all possible entities according to the query. From a representation learning perspective, this requires us to learn a conditional representation function $f(e|\mathcal{C}, \mathcal{R}_\mathcal{Q}, \mathcal{G})$ for each entity $e \in \mathcal{E}$. Transductive methods learn a shallow embedding for each answer entity $e \in \mathcal{E}$, and, therefore, cannot generalize to unseen entities. For inductive methods, the function $f(e|\mathcal{C}, \mathcal{R}_\mathcal{Q}, \mathcal{G})$ should generalize to some unseen answer entity e' (or unseen constant entity $c' \in \mathcal{C}'$) at inference time. Here, we discuss two solutions for devising such an inductive function.

The first solution is to **parameterize the representation of each entity** e **as a function of an invariant vocabulary** of *tokens* that does not change at training and inference. Particularly, the vocabulary might consist of unique relation types $\mathcal{R}$ that are always the same for $\mathcal{G}_{train}$ and $\mathcal{G}_{inf}$, and we are able to infer the representation of an unseen answer entity (or an unseen constant entity) as a function of its incident relations (cf. Fig. 2 left). The idea has been studied in NodePiece [12] for simple link prediction. Here, we adopt a similar idea to learn inductive entity representations for complex query answering. Once we obtain the representations for unseen entities, we can use any off-the-shelf decoding method (e.g., CQD-Beam [5]) for predicting the answer to the complex query. We denote this strategy as NodePiece-QE.

The second solution is to **parameterize** $f(e|\mathcal{C}, \mathcal{R}_\mathcal{Q}, \mathcal{G})$ **as a function of the relational structure**. Intuitively, an answer of a complex query can be decided solely based on the relational structure between the query constants and the answer (Fig. 1). Even after anonymizing entity names (and, hence, not learning any explicit entity embedding), we are still able to infer `Princeton` as an answer since it forms a distinctive relational structure with the query constants and conforms to the query structure. Similarly, intermediate nodes will be deemed correct if they follow a relational structure . In other words, we do not need to know the answer node is `Princeton`, but only need to know the relative position of `Princeton` w.r.t. the constants like `Nobel Prize` and `USA`. Based on this idea, we design $f(e|\mathcal{C}, \mathcal{R}_\mathcal{Q}, \mathcal{G})$ to be a relational structure search function. Such an idea has been studied in Neural Bellman-Ford Networks (NBFNet) [39] to search for a single relation in simple link prediction. Here, we chain several NBFNet instances with differentiable logic operations to learn inductive complex query in an end-to-end fashion. We denote this strategy as NBFNet-QE.

4.1 NodePiece-QE: Inductive Node Representation

Here, we aim at reconstructing node representations for seen and new entities without learning shallow node embedding vectors. To this end, we employ NodePiece [12], a compositional tokenization approach that learns an invariant vocabulary of *tokens* shared between training and inference graphs. Formally, given a vocabulary of tokens $t_i \in T$, each entity e_i is deterministically hashed into a set of representative tokens $e_i = [t_1, \ldots, t_k]$. An entity vector $\boldsymbol{e}_i$ is then obtained as a function of token embeddings $\boldsymbol{e}_i = f_\theta([\boldsymbol{t}_i, \ldots, \boldsymbol{t}_k]), \boldsymbol{t}_i \in \mathbf{T}^{|T| \times d}$ where the encoder function $f_\theta : \mathbb{R}^{k \times d} \to \mathbb{R}^d$ is parameterized with a neural network θ.

Since the set of relation types $\mathcal{R}$ is invariant for training and inference graphs, we can learn relation embeddings $\mathbf{R}^{|\mathcal{R}| \times d}$ and our vocabulary of learnable tokens T is comprised of distinct relation types such that entities are hashed into a set of unique incident relation types. For example (cf. Fig. 2 left), a middle node from a training graph $\mathcal{G}_{train}$ is hashed with a set of relations $e_i = [$↓↓↑$]$ that stands for two unique incoming relations ↓↓ and one unique outgoing relation ↑ . Passing the hashes through f_θ, we can reconstruct the whole entity embedding matrix $\mathbf{E}^{|\mathcal{E}_{train}| \times d}$. Additionally, it is possible to enrich entity and relation embeddings by passing them through a relational GNN encoder [30] over a target graph $\mathcal{G}$: $\mathbf{E}', \mathbf{R}' = \text{GNN}(\mathbf{E}, \mathbf{R}, \mathcal{G})$. In both ways, the entity embedding matrix $\mathbf{E}$ encodes a *joint* probability distribution $p(h, r, t)$ for all triples in a graph.

Having a uniform featurization mechanism for both seen and unseen entities, it is now possible to apply any previously-transductive complex query answering model with learnable entity embeddings and logical operators [22, 10, 23, 7]. Moreover, it was recently shown [5] that a combination of simple link prediction pre-training and a non-parametric logical executor allows to effectively answer complex FOL queries in the *inference-only* regime without training on any complex query sample. We adopt this Continuous Query Decomposition algorithm with beam search (CQD-Beam) as the main query answering decoder. CQD-Beam relies only on entity and relation embeddings $\mathbf{E}, \mathbf{R}$ pre-trained on a simple *1p* link prediction task. Then, given a complex query, CQD-Beam applies *t-norms* and *t-conorms* [18] that execute conjunctions ($\wedge$) and disjunctions ($\vee$) as non-parametric algebraic operations in the embedding space, respectively.

In our inductive setup (Fig. 2), we train a NodePiece encoder f_θ and relation embeddings $\mathbf{R}$ (and optionally a GNN) on the *1p* link prediction task over the training graph $\mathcal{G}_{train}$. We then apply the learned encoder to materialize entity representations of the inference graph $\mathbf{E}^{|\mathcal{E}_{inf}| \times d}$ and send them to CQD-Beam that performs a non-parametric decoding of complex FOL queries over new inference entities. The inference-only nature of NodePiece-QE is designed to be challenging and probing the abilities for zero-shot generalization in performing complex logical reasoning over larger graphs.

4.2 NBFNet-QE: Inductive Relational Structure Representation

The second strategy relies on learning inductive relational structure representations instead of explicit node representations. Having the same set of relation types $\mathcal{R}$ at training and inference time, we can parameterize each entity based on the relative relational structure between it and the anchor nodes in a given query. For instance (Fig. 2 right), given a query with a particular relational structure ≻→ and a set of anchor nodes, the representation of each node captures its relational structure relative to the anchor nodes. Each neighborhood expansion step is equivalent to the *projection* step. In our example, immediate neighboring nodes will capture the intersection pattern ≻ , and further nodes, in turn, capture the extended *intersection-projection* structure ≻→ .

Therefore, a node is likely to be an answer if its captured (or predicted) relational structure conforms with the query relational structure. As long as the set of relations is fixed, relation *projection* is performed in the same way for training or new unseen nodes. The idea of a one-hop (*1p*) projection for simple link prediction has been proposed by Neural Bellman-Ford Networks (NBFNet) [39].

In particular, given a relation *projection* query $(h, r, ?)$, NBFNet assigns unique initial states $\boldsymbol{h}^{(0)}$ to all nodes in a graph by applying an indicator function $\boldsymbol{h}_e^{(0)} = \textsc{Indicator}(h, v, r)$, i.e., a head node h is initialized with a learnable relation embedding $\boldsymbol{r}$ and all other nodes are initialized with zeros. Then, NBFNet applies L relational message passing GNN layers where each layer l has its own learnable relation embedding matrix $\mathbf{R}_l$ obtained as a projection of the initial relation $\mathbf{R}_l = \mathbf{W}_l \boldsymbol{r} + \boldsymbol{b}_l$. Final layer representations $\boldsymbol{h}^{(L)}$ are passed through an MLP and activation function σ to get a probability distribution over all nodes in a graph $p(t|h, r) = \sigma(\text{MLP}(\boldsymbol{h}^{(L)}))$. As each query spawns a uniquely initialized graph and message passing procedure, NBFNet is seen to be applying a *labeling trick* [35] to model a conditional probability distribution $p(t|h, r)$ which is provably more expressive than a joint distribution $p(h, r, t)$ produced by standard graph encoders.

Applied to complex queries, chaining k NBFNet instances allows to answer k-hop projection queries, e.g., two instances for *2p* queries. NBFNet-QE employs NBFNet as a *trainable* projection operator and endows it with differentiable, non-parametric *product* logic for modeling conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) over the *fuzzy sets* of all entities $\boldsymbol{x} \in [0,1]^{\mathcal{E}}$, i.e., after applying a logical operator (discussed in Appendix A), each entity's degree of truth is associated with a scalar in range $[0,1]$. For the next hop projection, the indicator function initializes a node state with a relation vector $\boldsymbol{r}_i$ weighted by a scalar probability predicted in the previous hop x_e: $\boldsymbol{h}_e^{(0)} = x_e \boldsymbol{r}_i$. Differentiable logical operators allow training NBFNet-QE end-to-end on complex queries.

5 Experiments

We designed the experimental agenda to demonstrate that inductive representation strategies are able to: (1) answer complex logical queries over new, unseen entities at inference time, i.e., when query anchors are new nodes (Section 5.2); (2) predict new correct answers for known *training* queries when executed over larger inference graphs, i.e., when query anchors come from the training graph but variables and answers belong to the larger inference graph (Section 5.3); (3) generalize to inference graphs of up to 500% larger than training graphs; (4) scale to inductive query answering over graphs of millions of nodes when updated with 500k new nodes and 5M new edges (Section 5.4).

5.1 Setup & Dataset

Dataset. Due to the absence of inductive logical query benchmarks, we create a novel suite of datasets[4] based on FB15k-237 [28] (open license) and following the BetaE [23] query sampling methodology. Given a source graph with $\mathcal{E}$ entities, we sample $|\mathcal{E}_{train}| = r \cdot |\mathcal{E}|, r \in [0.1, 0.9]$ nodes to induce a training graph $\mathcal{G}_{train}$. For validation and test graphs, we split the remaining set of entities into two non-overlapping sets each with $\frac{1-r}{2}|\mathcal{E}|$ nodes. We then merge training and unseen nodes into the inference set of nodes $\mathcal{E}_{inf}$ and induce inference graphs for validation and test from those sets, respectively, i.e., $\mathcal{E}_{inf}^{val} = \mathcal{E}_{train} \cup \mathcal{E}_{val}$ and $\mathcal{E}_{inf}^{test} = \mathcal{E}_{train} \cup \mathcal{E}_{test}$. That is, validation and test inference graphs both extend the training graph but their sets of new entities are disjoint. Finally, we sample and remove 15% of edges $\mathcal{T}_{\text{pred}}$ in the inference graphs as missing edges for link prediction pre-training and query sampling. Overall, we sample 9 such datasets varying r to obtain ratios of inference graph size to the training graph $\mathcal{E}_{inf}/\mathcal{E}_{train}$ from 105% to 550%.

For each dataset, we employ the query sampler from BetaE [23] to extract 14 typical query types *1p/2p/3p/2i/3i/ip/pi/2u/up/2in/3in/inp/pin/pni*. Training queries are sampled from the training graph $\mathcal{G}_{train}$, validation and test queries are sampled from their respective inference graphs $\mathcal{G}_{inf}$ where at least one edge belongs to $\mathcal{T}_{\text{pred}}$ and has to be predicted at inference time.

As inference graphs extend training graphs, training queries are very likely to have new answers being executed over $\mathcal{G}_{inf}$ with simple graph traversal and without any link prediction. We create an additional set of true answers for all training queries executed over the test inference graph $\mathcal{G}_{inf}^{test}$ to measure the generalization capabilities of query answering models. This is designed to be an inference task and extends the *faithfullness* experiment [25]. Dataset statistics can be found in Appendix B.

Evaluation Protocol. Following the literature [23], query answers are separated into two sets: *easy answers* that only require graph traversal over existing edges, and *hard answers* that require inferring missing links to achieve the answer node. For the main experiment, evaluation involves ranking of *hard* answers against all entities having easy ones filtered out. For evaluating training queries on inference graphs, we only have *easy* answers and rank them against all entities. We report Hits@10 as the main performance metric on different query types.

Implementation Details. All NodePiece [12]-based models were pre-trained until convergence on a simple *1p* link prediction task with the relations-only vocabulary and entity tokenization, MLP encoder, and ComplEx [29] scoring function. We used a 2-layer CompGCN [30] as an optional message passing encoder on top of NodePiece features. The non-parametric CQD-Beam [5] decoder for answering complex queries is tuned for each query type based on the validation set of queries, most of the setups employ a *product t-norm*, sigmoid entity score normalization, and beam size of 32. Following the literature, the NBFNet-QE models were trained on 10 query patterns (*1p/2p/3p/2i/3i/2in/3in/inp/pin/pni*) where *ip/pi/2u/up* are only seen at inference time. Each model employs a 4-layer NBFNet [39] as a trainable projection operator with DistMult [32] composi-

[4] Available to reviewers, will be published under the CC0 license

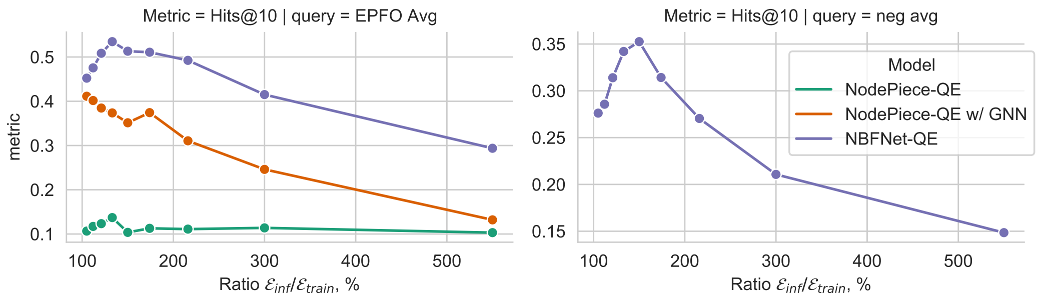

Figure 3: Aggregated Hits@10 performance of **test queries** (involving unseen entities) executed on inference graphs of different ratios compared to training graphs. NodePiece-based models are *inference-only* and support EPFO queries, NBFNet-QE is trainable and supports negation queries.

Table 1: Test Hits@10 results (%) on answering inductive FOL queries when $\mathcal{E}_{inf}/\mathcal{E}_{train} = 175\%$. avg$_p$ is the average on EPFO queries ($\wedge$, $\vee$). avg$_n$ is the average on queries with negation.

Model	**avg$_p$**	**avg$_n$**	**1p**	**2p**	**3p**	**2i**	**3i**	**pi**	**ip**	**2u**	**up**	**2in**	**3in**	**inp**	**pin**	**pni**
BetaE *(transductive)*	1.8	0.4	2.8	0.8	0.4	3.2	5.1	2.1	1.1	0.3	0.3	0.3	0.7	0.4	0.3	0.2
Inductive Inference-only																
NodePiece-QE	11.3	-	24.1	10.5	10.0	11.0	12.8	9.1	8.6	7.3	8.3	-	-	-	-	-
NodePiece-QE w/ GNN	37.4	-	56.5	27.1	16.0	49.3	57.1	35.7	31.8	39.7	23.6	-	-	-	-	-
Inductive Trainable																
NBFNet-QE	51.1	31.4	66.1	40.9	31.2	73.0	83.3	58.3	41.3	37.8	27.8	31.1	44.3	28.4	25.2	28.0

tion function and PNA [9] aggregation. Other logical operators ($\wedge, \vee, \neg$) are executed with the non-parametric *product t-norm*. Both NodePiece-QE and NBFNet-QE are implemented[5] with PyTorch [21] and trained with the Adam [17] optimizer. NodePiece-QE models were pre-trained and evaluated on a single Tesla V100 32 GB GPU whereas NBFNet-QE models were trained and evaluated on 4 Tesla V100 16GB. All hyperparameters are listed in Appendix D.

5.2 Complex Query Answering over Unseen Entities on Differently Sized Inference Graphs

First, we probe *inference-only* NodePiece-based embedding models and *trainable* NBFNet-QE in the inductive setup, i.e., query answering over unseen nodes requiring link prediction over unseen nodes. Table 1 summarizes the results on a reference dataset with ratio $\mathcal{E}_{inf}/\mathcal{E}_{train}$ of 175% while Fig. 3 illustrates a bigger picture on all datasets (we provide a detailed breakdown by query type for all splits in Appendix C). We observe that even inference-only models pre-trained solely on simple *1p* link prediction exhibit non-trivial performance in answering queries with unseen entities. Paired with an additional GNN encoder, the inference-only baseline exhibits significantly better performance over all query types and inference graphs up to 300% larger than training graphs.

The trainable NBFNet-QE models expectedly outperform non-trainable baselines and can tackle queries with negation ($\neg$). Here, we confirm that the *labeling trick* [35] and conditional $p(t|h, r)$ modeling better capture the relation projection problem than joint $p(h, r, t)$ encoding approaches.

Still, all evaluated models with message passing, both inference-only NodePiece-QE with GNN and trainable NBFNet-QE, suffer from increasing the size of the inference graph and having more unseen entities. Reaching best results on $\mathcal{E}_{inf}/\mathcal{E}_{train}$ ratios around 130%, both approaches steadily deteriorate up until final 550% by 20 absolute Hits@10 points on EPFO queries and negation queries. We attribute this deterioration to the known generalization issues [19, 34] of message passing GNNs when performing inference over much larger graph than the network has seen during training. On the other hand, a simple NodePiece-QE model without message passing retains similar performance independently of the inference graph size.

Lastly, we observe that lower performance of inference-only NodePiece models can be also attributed to underfitting (cf. train graph charts in Fig. 4). Although *1p* link predictors were trained until

[5] Source code is available to reviewers

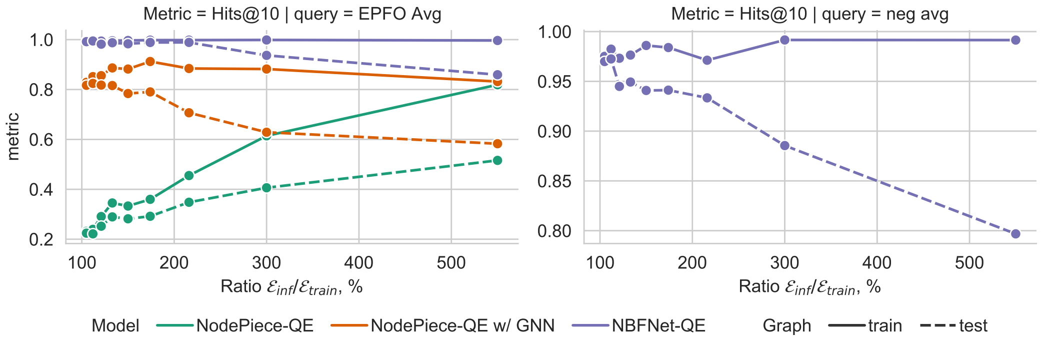

Figure 4: Aggregated Hits@10 performance of **training queries** on the original training and extended test inference graphs where queries have new correct answers. NodePiece-based models are *inference-only* and support EPFO queries, NBFNet-QE is trainable and supports negation queries.

convergence (on the inductive validation set of missing triples), the performance of training queries on training graphs with *easy answers* that require only relation traversal without predicting missing edges is not yet saturated. This fact suggests that better fitting entity featurization (obtained by NodePiece or other strategies) could further improve the test performance in the inference-only regime. We leave the search of such strategies for future work.

5.3 Predicting New Answers for Training Queries on Larger Inference Graphs

Simulating the incremental addition of new edges in graph databases, we evaluate the performance of our inference-only and trainable QE models on *training* queries on the original training graph and extended inference graph (with added test edges). As databases are able to immediately retrieve new answers to known queries after updating the graph, we aim at exploring and quantifying this behaviour of neural reasoning models. In this experiment, we probe training queries and their *easy answers* that require performing only graph traversal without predicting missing links in the inference graph. While execution of training queries over the *training* graph indicates how well the model could fit training data, executing training queries over the bigger *inference* graph with new entities aims to capture basic reasoning capabilities of QE models in the inductive regime.

Particular challenges arising when executing training queries over a bigger graph are: (1) the same queries can have more correct answers as more new nodes and edges satisfying the query pattern might have been added (as in Fig. 1); (2) more new entities create a “distractor” setting with more false positives. Generally, evaluation of training queries on the inference graph can be considered as an extended version of the *faithfullness* [25] evaluation that captures how well a trained model can answer original training queries, i.e., memorization capacity. In all 9 datasets, most of training queries have at least one new correct answer in the inference graph (more details in Appendix B).

Fig. 4 illustrates the performance of the inference-only NodePiece-QE (without and with GNN) and trainable NBFNet-QE. Generally, NBFNet-QE fits the training query data almost perfectly confirming the original finding [39] that NBFNet can perform graph traversal akin to symbolic rule-based models. NBFNet-QE can also find new correct answers on graphs up to 300% larger than training ones. Then, the performance quickly deteriorates which we attribute to the *distractor* factor with more unseen entities and the already mentioned generalization issue on larger inference graphs.

The inference-only NodePiece-QE models, as expected, do not fully fit the training data as they were never trained on complex queries. Still, the inference-only models exhibit non-trivial performance in finding more answers on graphs up to 200% larger than training ones with relatively small performance margins compared to training queries. The most surprising observation is that GNN-free NodePiece-QE models improve the performance on both training and inference graphs as the graphs (and the $\mathcal{E}_{inf}/\mathcal{E}_{train}$ ratio) grow larger while GNN-enriched models steadily deteriorate. We attribute this growth to the relation-based NodePiece tokenization and its learned features that tend to be more discriminative in larger inference graphs where new nodes have smaller degree and thus can be better identified by their incident relation types. We provide more experimental results for each dataset ratio with breakdown by query type in Appendix C.

5.4 Scaling to Millions of Nodes on WikiKG-QE

Finally, we perform a scalability experiment evaluating complex query answering in the inductive mode on a new large dataset *WikiKG-QE* constructed from OGB WikiKG 2 [15] (CC0 license). While the original task is transductive link prediction, we split the graph into a training graph of 1.5M entities (5.8M edges, 512 unique relation types) and validation (test) graphs of 500k unseen nodes (5M known and 600k missing edges) each. The resulting validation (test) inference graphs are therefore of 2M entities and 11M edges with the $\mathcal{E}_{inf}/\mathcal{E}_{train}$ ratio of 133% (details are in Appendix B).

None of GNN-enabled models can scale to such sizes, so we use a basic inference-only NodePiece-QE. Due to the problem size, we only sample 10k EPFO queries of each type from the *test inference* graph to run in the inference-only regime. Each query has at least one missing edge to be predicted at inference. The answers are ranked against all 2M entities in the filtered setting (in contrast to the OGB task that ranks against 1000 pre-computed negative samples) and Hits@100 as the target metric.

We pre-train a NodePiece encoder (in addition to relation types, we tokenize nodes with a vocabulary of 20k nodes, total 3M parameters in the encoder) with the ComplEx decoder on *1p* link prediction over the training graph for 1M steps (see Appendix D for hyperparameters). Then, the graph is extended with 500k new nodes and 5M new edges forming the inference graph. Then, using the pre-trained encoder, we materialize representations of entities (both seen and new) and relations from this inference graph. Finally, CQD-Beam executes the queries against the bigger inference graph extended with 500k new nodes and 5M new edges.

Table 2: Test Hits@100 of NodePiece-QE on WikiKG-QE (2M nodes, 11M edges including 500k new nodes and 5M new edges) in the inference-only regime. avg$_p$ is the average on EPFO queries.

Model	**avg$_p$**	**1p**	**2p**	**3p**	**2i**	**3i**	**pi**	**ip**	**2u**	**up**
NodePiece-QE	9.2	22.6	5.2	3.9	11.6	17.4	7.0	4.5	7.4	3.2
NodePiece-QE w/ GNN	10.1	66.6	0.9	0.6	5.4	8.2	2.3	0.8	5.2	0.5

As shown in Table 2, we find a non-trivial performance of the inference-only model on EPFO queries demonstrating that inductive *node representation* QE models are able to scale to graphs with hundreds of thousands of new nodes and millions of new edges in the zero-shot fashion. That is, answering complex queries over unseen entities is available right upon updating the graph without the need to retrain a model. This fact paves the way for the concept of *neural graph databases* capable of performing zero-shot inference over updatable graphs without expensive retraining.

6 Limitations and Future Work

Limitations. With the two proposed inductive query answering strategies, we observe a common trade-off between the performance and computational complexity. That is, inductive *node representation* models like NodePiece-QE are fast, scalable, and can be executed in the inference-only regime but underperform compared to the inductive *relational structure representation* models like NBFNet-QE. On the other hand, NBFNet-QE incurs high computational costs due to executing each query on a uniquely initialized graph instance. Alleviating this issue is a key to scalability.

Societal Impact. The inductive setup assumes running inference on (partly) unseen data, that is, the nature of this unseen data might be out-of-distrbution, unknown and potentially malicious. This fact has to be taken into account when evaluating predictions and overall system trustworthiness.

Conclusion and Future Work. In this work, we defined the problem of inductive complex logical query answering and proposed two possible parameterization strategies based on *node* and *relational structure* representations to deal with new, unseen entities at inference time. Experiments demonstrated that both strategies are able to answer complex logical queries over unseen entities as well as identify new answers on larger inference graphs. In the future work, we plan to extend the inductive setup to completely disjoint training and inference graphs, expand the set of supported logical query patterns aligned with popular queries over real-world KGs, enable reasoning over continuous features like texts and numbers, support more KG modalities like hypergraphs and hyper-relational graphs, and further explore the concept of neural graph databases.

References

[1] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[2] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

[3] Dimitrios Alivanistos, Max Berrendorf, Michael Cochez, and Mikhail Galkin. Query embedding on hyper-relational knowledge graphs. In *International Conference on Learning Representations*, 2022.

[4] Alfonso Amayuelas, Shuai Zhang, Xi Susie Rao, and Ce Zhang. Neural methods for logical reasoning over knowledge graphs. In *International Conference on Learning Representations*, 2022.

[5] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations*, 2021.

[6] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013.

[7] Xuelu Chen, Ziniu Hu, and Yizhou Sun. Fuzzy logic based logical query answering on knowledge graph. In *International Conference on Machine Learning*. PMLR, 2021.

[8] Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K. Reddy. Probabilistic entity representation model for reasoning over knowledge graphs. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[9] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[10] Daniel Daza and Michael Cochez. Message passing query embedding. *arXiv preprint arXiv:2002.02406*, 2020.

[11] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[12] Mikhail Galkin, Etienne Denis, Jiapeng Wu, and William L. Hamilton. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. In *International Conference on Learning Representations*, 2022.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[14] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. *Advances in Neural Information Processing Systems*, 31, 2018.

[15] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference*

on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[16] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[18] Erich-Peter Klement, Radko Mesiar, and Endre Pap. Triangular norms. position paper I: basic analytical and algebraic properties. *Fuzzy Sets Syst.*, 143(1):5–26, 2004.

[19] Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4204–4214, 2019.

[20] Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. Answering complex queries in knowledge graphs with bidirectional sequence encoders. *CoRR*, abs/2004.02596, 2020.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[22] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations*, 2019.

[23] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

[24] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

[25] Haitian Sun, Andrew O. Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W. Cohen. Faithful embeddings for knowledge base queries. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[26] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[27] Komal K. Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9448–9457. PMLR, 2020.

[28] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics.

[29] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[30] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[31] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 515–526. ACM, 2014.

[32] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[33] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2319–2328, 2017.

[34] Gilad Yehudai, Ethan Fetaya, Eli A. Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11975–11986. PMLR, 2021.

[35] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9061–9073. Curran Associates, Inc., 2021.

[36] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2731–2741, 2019.

[37] Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems*, 34, 2021.

[38] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, Chang Ma, Runcheng Liu, Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Torchdrug: A powerful and flexible machine learning platform for drug discovery, 2022.

[39] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and sample data are included in the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Dataset creation process is described in Section 5.1 with more details in Appendix B. Hyperparameters are specified in Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We observe negligible variance w.r.t. random seeds
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Training details are specified in Section 5.1 and in Appendix D.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Due to the overall size, we include a sample of the benchmarking suite in the supplemental material and will openly publish the whole dataset.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No personal data involved
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] The datasets are anonymized, we discuss it in Appendix B.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Differentiable Logical Operators

T-norms ($\top$) and t-conorms ($\perp$) are *fuzzy* versions of conjunction ($\wedge$) and disjunction ($\vee$), respectively. Fuzzy operators can be applied to vectors of continuous values within a certain range, e.g., $[0,1]^d$, depending on the chosen *fuzzy logic*, and are executed as algebraic operations which makes them differentiable. Different *fuzzy logics* implement different t-norms and t-conorms. In this work, we experiment with two such logics: *product logic* and *Gödel (min) logic*. In the product logic, conjunction $\mathcal{C}$, disjunction $\mathcal{D}$, and negation $\mathcal{N}$ are modeled as follows:

$$\begin{aligned}
\mathcal{C}(\boldsymbol{x},\boldsymbol{y}) &= \boldsymbol{x} \odot \boldsymbol{y} \\
\mathcal{D}(\boldsymbol{x},\boldsymbol{y}) &= \boldsymbol{x} + \boldsymbol{y} - \boldsymbol{x} \odot \boldsymbol{y} \\
\mathcal{N}(\boldsymbol{x}) &= \mathbf{1} - \boldsymbol{x}
\end{aligned}$$

where inputs $\boldsymbol{x}, \boldsymbol{y} \in [0,1]^d$ are d-dimensional vectors with values in the range $[0,1]$, $\odot$ is the element-wise multiplication, and $\mathbf{1}$ is the *universe* vector of all ones.

In the Gödel logic, conjunction $\mathcal{C}$ and disjunction $\mathcal{D}$ are modeled as *min* and *max*, respectively:

$$\begin{aligned}
\mathcal{C}(\boldsymbol{x},\boldsymbol{y}) &= \mathit{min}(\boldsymbol{x},\boldsymbol{y}) \\
\mathcal{D}(\boldsymbol{x},\boldsymbol{y}) &= \mathit{max}(\boldsymbol{x},\boldsymbol{y})
\end{aligned}$$

For NBFNet-QE we employ solely the product logic for end-to-end training on all types of complex queries. For NodePiece-QE and its inference-only mechanism based on CQD-Beam, we may select the best performing logic for each query type based on the validation set. The chosen operators for NodePiece-QE are reported in Table 12 in Appedix D.

B Benchmarking Datasets Details

We sampled 9 datasets (used in Section 5.2 and Section 5.3) from the original FB15k-237 [28] with already added inverse edges for ensuring reachability and connectedness of the underlying graph for the subsequent query sampling. Creation details are provided in the Section 5.1 and statistics on the sampled graphs are presented in Table 3. Varying the ratio of entities in the inference graph to the training graph $\mathcal{E}_{inf}/\mathcal{E}_{train}$, we aim at measuring inductive capabilities of proposed strategies in the out-of-distribution size generalization scenario. To measure scalability of inductive query answering approaches, we create WikiKG-QE, an inductive split of the originally transductive OGB WikiKG 2 [15], following the same sampling strategy as for 9 Freebase datasets.

Table 3: Sampled graphs statistics for various ratios $\mathcal{E}_{inf}/\mathcal{E}_{train}$. Originally inverse triples are included in all graphs except WikiKG-QE. $\mathcal{R}$ - number of unique relation types, $\mathcal{E}$ - number of entities in various splits, $\mathcal{T}$ - number of triples. Validation and Test splits contain an inference graph ($\mathcal{E}_{inf}$, $\mathcal{T}_{inf}$) which is a superset of the training graph with new nodes, and missing edges to predict $\mathcal{T}_{pred}$.

Ratio, %	$\mathcal{R}$	$\mathcal{E}_{total}$	Training Graph		Validation Graph			Test Graph		
			$\mathcal{E}_{train}$	$\mathcal{T}_{train}$	$\mathcal{E}^{val}_{inf}$	$\mathcal{T}^{val}_{inf}$	$\mathcal{T}^{val}_{pred}$	$\mathcal{E}^{test}_{inf}$	$\mathcal{T}^{test}_{inf}$	$\mathcal{T}^{test}_{pred}$
106%	472	14,480	13,032	492,334	13,756	541,237	8,631	13,756	548,469	9,907
113%	466	14,463	11,567	392,672	13,015	486,517	16,557	13,015	479,993	15,403
122%	460	14,416	10,079	294,556	12,247	417,213	21,643	12,248	420,110	22,152
134%	458	14,271	8,528	218,192	11,397	361,228	25,240	11,402	364,747	25,861
150%	450	14,271	7,085	153,280	10,679	317,862	29,044	10,677	309,554	27,578
175%	438	13,960	5,429	98,664	9,707	258,788	28,236	9,682	268,987	30,051
217%	442	13,864	3,969	55,052	8,935	230,509	30,949	8,898	211,947	27,683
300%	402	13,992	2,643	24,338	8,322	190,655	29,337	8,313	174,442	26,482
550%	346	13,522	1,057	5,324	7,304	136,898	23,210	7,275	139,093	23,583
					WikiKG-QE					
133%	512	2,492,122	1,494,033	5,824,868	1,992,739	9,466,319	638,389	1,993,416	10510906	824,713

In all datasets, entities and relations are anonymized and only have an integer ID. Furthermode, inference graphs at validation and test time are supersets of the respective training graph with new nodes and edges. The amount of new unique nodes is simply the difference $\mathcal{E}_{inf} - \mathcal{E}_{train}$ between entities in those graphs, e.g., for the dataset of ratio 175%, the validation inference graph contains $4,278$ new nodes and test inference graph contains $4,253$ news nodes. Note that those $4,278$ and $4,253$ nodes are unique for each graph and do not overlap. That is, validation inference and test inference graphs are disconnected except sharing the same core training graph.

Then, for each created inductive dataset, we sample queries of 14 query patterns following the BetaE [23] procedure. That is, *training* queries are sampled from the *training* graph $\mathcal{G}_{train}$ and have only *easy* answers reachable by simple edge traversal. Validation and test queries are sampled from the respective splits, e.g., *validation* queries are sampled from the validation graph $\mathcal{G}_{val}$ using entities from the validation inference graph $\mathcal{E}_{inf}^{val}$ (which, in turn, are a union of training nodes and new, unseen validation nodes $\mathcal{E}_{train} \cup \mathcal{E}_{val}$), and at least one edge in each query belongs to $\mathcal{T}_{pred}^{val}$ and has to be predicted during query execution. Queries might have *easy* answers that are directly reachable by traversing edges $\mathcal{T}_{inf}^{val}$ in the validation inference graph, whereas *hard* answers are only reachable after predicting missing edges from the set $\mathcal{T}_{pred}^{val}$. Final evaluation metrics are computed only based on the *hard* answers. Following the literature [23], we only retain queries that have less than 1000 answers. Table 4 summarizes the statistics on the sampled queries for each dataset ratio, each graph, and query type that we use in Section 5.2 for evaluating inductive query answering performance. In graphs with smaller inference graphs and smaller number of missing triples, we sample fewer queries with negation (*2in, 3in, inp, pin, pni*) for validation and test splits. For WikiKG-QE, due to its size, we only sample 10k EPFO queries to be executed in the inference-only regime without training (at the moment, CQD-Beam does not support queries with negation). We use those queries in Section 5.4 to evaluate scalability of NodePiece-QE and prediction quality in the inference-only mode.

Table 4: Statistics on sampled queries for each dataset ratio and query type. For WikiKG-QE, we only sample EPFO queries without negation.

Ratio	Graph	**1p**	**2p**	**3p**	**2i**	**3i**	**pi**	**ip**	**2u**	**up**	**2in**	**3in**	**inp**	**pin**	**pni**
106%	training	134,988	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	5,753	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	1,000	1,000	1,000	1,000	1,000
	test	6,370	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	1,000	1,000	1,000	1,000	1,000
113%	training	111,869	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	10,513	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	1,000	1,000	1,000	1,000	1,000
	test	10,090	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	1,000	1,000	1,000	1,000	1,000
122%	training	90,648	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	13,656	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	5,000	5,000	5,000	5,000	5,000
	test	13,833	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	5,000	5,000	5,000	5,000	5,000
134%	training	70,237	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	15,609	50,000	50,000	50,000	50,000	50,000	50,000	20,000	20,000	5,000	5,000	5,000	5,000	5,000
	test	15,768	50,000	50,000	50,000	50,000	50,000	50,000	20,000	20,000	5,000	5,000	5,000	5,000	5,000
150%	training	54,501	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	17,414	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	5,000	5,000	5,000	5,000	5,000
	test	16,733	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	5,000	5,000	5,000	5,000	5,000
175%	training	38,573	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	40,000	50,000	50,000	50,000
	validation	17,054	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
	test	17,867	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
217%	training	22,457	30,000	30,000	50,000	50,000	50,000	50,000	50,000	50,000	30,000	30,000	50,000	50,000	50,000
	validation	18,154	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
	test	16,790	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
300%	training	11,920	15,000	15,000	40,000	40,000	50,000	50,000	50,000	50,000	15,000	15,000	50,000	40,000	50,000
	validation	17,387	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
	test	16,014	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
550%	training	3,253	15,000	15,000	40,000	40,000	50,000	50,000	50,000	50,000	10,000	10,000	30,000	30,000	30,000
	validation	14,027	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
	test	14,183	50,000	50,000	50,000	50,000	50,000	50,000	50,000	50,000	10,000	10,000	10,000	10,000	10,000
									WikiKG-QE						
133%	training	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	-	-	-	-	-
	validation	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	-	-	-	-	-
	test	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	-	-	-	-	-

Furthermore, for the experiment in Section 5.3 to measure the abilities of inductive models to find new answers of known queries, we take the created **training** queries and find their *easy* answers in the validation inference $\mathcal{G}_{inf}^{val} = (\mathcal{E}_{inf}^{val}, \mathcal{T}_{inf}^{val})$ and test inference $\mathcal{G}_{inf}^{test} = (\mathcal{E}_{inf}^{test}, \mathcal{T}_{inf}^{test})$ graphs. That is, those new answers do not require predicting missing edges in the inference graphs and only require a model to execute edge traversal to find (if any) new correct answers involving new, unseen entities and edges. For the validation (test) split, we only count such training queries q whose answer set in

Table 5: Statistics on **training** EPFO queries that have a different (often, larger) answer set when executed against validation and test inference graphs. We list the original number of training queries, number of those queries with new *easy* answers in the validation (In val) and test graphs (In test), as well as their percentage ratio to the total number. Most queries (except *2i,3i*) have new answer sets.

Ratio	Graph	1p		2p		3p		2i		3i		pi		ip		2u		up	
		#Q	%	#Q	%	#Q	%	#Q	%	#Q	%	#Q	%	#Q	%	#Q	%	#Q	%
106%	Train	134,988	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	11,834	8.8	30,541	61.1	40,466	80.9	7,414	14.8	4,605	9.2	15,410	30.8	28,846	57.7	33,069	66.1	41,483	83.0
	In test	13,824	10.2	30,481	61.0	39,995	80.0	7,311	14.6	4,315	8.6	15,225	30.5	28,782	57.6	33,068	66.1	40,909	81.8
113%	Train	111,869	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	18,048	16.1	34,773	69.5	42,955	85.9	10,163	20.3	5,326	10.7	18,583	37.2	32,986	66.0	37,570	75.1	44,209	88.4
	In test	17,271	15.4	34,666	69.3	42,907	85.8	10,149	20.3	5,384	10.8	18,916	37.8	33,245	66.5	37,589	75.2	44,067	88.1
122%	Train	90,648	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	20,253	22.3	37,454	74.9	44,415	88.8	11,850	23.7	6,128	12.3	21,787	43.6	35,921	71.8	40,396	80.8	45,660	91.3
	In test	21,077	23.3	37,593	75.2	44,479	89.0	11,976	24.0	6,036	12.1	21,919	43.8	35,969	71.9	40,467	80.9	45,754	91.5
134%	Train	70,237	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	19,360	27.6	37,446	74.9	44,421	88.8	12,600	25.2	7,094	14.2	21,975	44.0	36,598	73.2	41,371	82.7	45,597	91.2
	In test	20,241	28.8	38,135	76.3	45,002	90.0	13,036	26.1	7,268	14.5	22,432	44.9	37,215	74.4	41,968	83.9	45,888	91.8
150%	Train	54,501	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	18,302	33.6	39,579	79.2	45,704	91.4	14,312	28.6	7,594	15.2	25,152	50.3	39,059	78.1	43,072	86.1	46,758	93.5
	In test	16,791	30.8	39,625	79.3	45,809	91.6	14,348	28.7	7,539	15.1	24,982	50.0	39,032	78.1	42,842	85.7	46,765	93.5
175%	Train	38,573	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	13,637	35.4	39,658	79.3	45,985	92.0	15,293	30.6	8,625	17.3	25,382	50.8	39,108	78.2	43,506	87.0	46,764	93.5
	In test	13,978	36.2	39,575	79.2	45,968	91.9	15,347	30.7	8,581	17.2	25,414	50.8	38,999	78.0	43,656	87.3	46,778	93.6
217%	Train	22,457	100.0	30,000	100.0	30,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	10,008	44.6	23,609	78.7	26,875	89.6	15,379	30.8	8,731	17.5	24,688	49.4	39,376	78.8	43,585	87.2	46,338	92.7
	In test	9,791	43.6	23,520	78.4	27,017	90.1	15,385	30.8	8,712	17.4	24,874	49.7	39,398	78.8	43,583	87.2	46,389	92.8
300%	Train	11,920	100.0	15,000	100.0	15,000	100.0	40,000	100.0	40,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	6,008	50.4	12,225	81.5	13,509	90.1	12,542	31.4	6,993	17.5	27,002	54.0	41,088	82.2	43,837	87.7	46,715	93.4
	In test	5,986	50.2	12,147	81.0	13,472	89.8	12,248	30.6	6,879	17.2	26,938	53.9	41,063	82.1	43,875	87.8	46,727	93.5
550%	Train	3,253	100.0	15,000	100.0	15,000	100.0	40,000	100.0	40,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	1,863	57.3	11,678	77.9	12,793	85.3	11,682	29.2	7,459	18.6	23,544	47.1	39,279	78.6	39,011	78.0	45,383	90.8
	In test	1,785	54.9	11,459	76.4	12,755	85.0	10,736	26.8	7,081	17.7	22,939	45.9	39,177	78.4	36,909	73.8	45,296	90.6

Table 6: Statistics on **training** negation queries that have a different (often, larger) answer set when executed against validation and test inference graphs. We list the original number of training queries, number of those queries with new *easy* answers in the validation (In val) and test graphs (In test), as well as their percentage ratio to the total number. Most queries have new answer sets.

Ratio	Graph	2in		3in		pin		pni		inp	
		#Q	%	#Q	%	#Q	%	#Q	%	#Q	%
106%	Train	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	25,212	50.4	18,851	37.7	37,619	75.2	26,859	53.7	37,448	74.9
	In test	25,395	50.8	18,677	37.4	37,298	74.6	26,825	53.7	36,868	73.7
113%	Train	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	30,812	61.6	23,642	47.3	41,676	83.4	32,247	64.5	41,833	83.7
	In test	31,035	62.1	23,620	47.2	41,695	83.4	32,351	64.7	41,687	83.4
122%	Train	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	35,066	70.1	27,325	54.7	44,149	88.3	36,253	72.5	44,085	88.2
	In test	35,451	70.9	27,548	55.1	44,290	88.6	36,565	73.1	44,207	88.4
134%	Train	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	37,592	75.2	29,415	58.8	45,189	90.4	38,261	76.5	45,098	90.2
	In test	37,948	75.9	29,925	59.9	45,444	90.9	38,447	76.9	45,631	91.3
150%	Train	50,000	100.0	40,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	40,368	80.7	26,354	65.9	46,630	93.3	40,992	82.0	46,446	92.9
	In test	40,354	80.7	26,823	67.1	46,612	93.2	40,997	82.0	46,553	93.1
175%	Train	50,000	100.0	40,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	42,265	84.5	29,046	72.6	47,325	94.7	42,361	84.7	47,328	94.7
	In test	42,224	84.4	29,392	73.5	47,293	94.6	42,480	85.0	47,438	94.9
217%	Train	30,000	100.0	30,000	100.0	50,000	100.0	50,000	100.0	50,000	100.0
	In val	26,402	88.0	22,665	75.6	47,905	95.8	43,705	87.4	47,987	96.0
	In test	26,412	88.0	22,562	75.2	48,019	96.0	43,736	87.5	47,854	95.7
300%	Train	15,000	100.0	15,000	100.0	50,000	100.0	40,000	100.0	50,000	100.0
	In val	13,612	90.7	11,980	79.9	48,663	97.3	36,046	90.1	48,570	97.1
	In test	13,549	90.3	11,690	77.9	48,502	97.0	36,091	90.2	48,461	96.9
550%	Train	10,000	100.0	10,000	100.0	30,000	100.0	30,000	100.0	30,000	100.0
	In val	9,353	93.5	8,573	85.7	29,480	98.3	27,764	92.5	29,087	97.0
	In test	9,169	91.7	8,389	83.9	29,295	97.7	27,298	91.0	29,127	97.1

this split is *different* from the answer set in the training graph, e.g., $\mathcal{A}_q^{val} \neq \mathcal{A}_q^{train}$. We summarize the statistics of identified new answer sets in all datasets in Table 5 (for EPFO queries) and Table 6 (for queries with negations). We find that in most query patterns across all dataset ratios, training queries indeed have new answer sets when executed against validation or test inference graphs.

C More Experimental Results

Here, we present a detailed breakdown of query answering performance measured in Sections 5.2 and 5.3 by query type. Fig. 5 and Table 7 contain detailed results from Section 5.2 of executing **test** queries with new, unseen entities over inference graphs of various ratios of new entities.

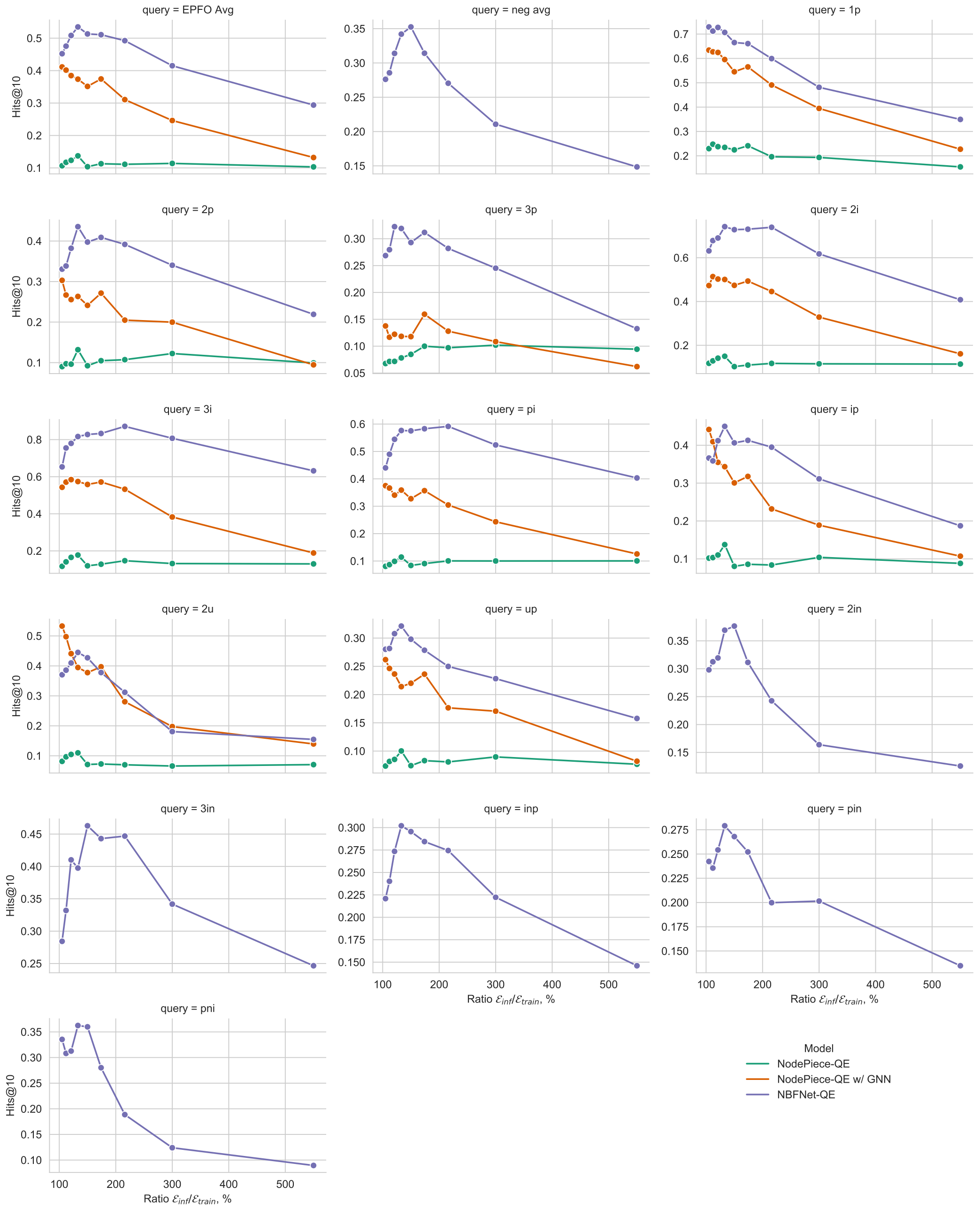

Figure 5: Hits@10 results on answering **test** inductive FOL queries on all ratios $\mathcal{E}_{inf}/\mathcal{E}_{train}$.

Table 7: Test Hits@3 and Hits@10 results (%) on answering **test** inductive FOL queries on all ratios $\mathcal{E}_{inf}/\mathcal{E}_{train}$. avg$_p$ is the average on EPFO queries ($\wedge$, $\vee$). avg$_n$ is the average on queries with negation.

Ratio	Model	Metric	**avg**$_p$	**avg**$_n$	1p	2p	3p	2i	3i	pi	ip	2u	up	2in	3in	inp	pin	pni
550%	NodePiece-QE	Hits@3	4.8		7.6	4.8	4.4	5.1	5.6	4.3	4.5	3.4	3.6					
		Hits@10	10.3		15.5	9.9	9.4	11.4	13.0	10.0	8.8	7.0	7.7					
	NodePiece-QE w/ GNN	Hits@3	6.8		11.5	4.9	3.0	8.0	9.5	6.0	5.9	8.1	4.4					
		Hits@10	13.2		22.7	9.5	6.2	16.1	18.9	12.6	10.7	13.9	8.2					
	NBFNet-QE	Hits@3	22.6	9.4	27.3	15.7	8.4	32.8	53.6	30.5	13.9	11.1	10.1	8.2	15.9	8.7	8.2	5.9
		Hits@10	29.4	14.8	35.0	21.9	13.3	40.8	63.2	40.3	18.7	15.5	15.8	12.6	24.7	14.6	13.5	8.9
300%	NodePiece-QE	Hits@3	5.4		11.2	6.2	4.9	4.9	5.4	4.1	4.7	2.9	4.5					
		Hits@10	11.4		19.3	12.2	10.2	11.5	13.2	10.0	10.4	6.6	9.0					
	NodePiece-QE w/ GNN	Hits@3	13.8		23.4	11.5	5.6	18.5	22.3	12.8	10.2	10.1	9.6					
		Hits@10	24.6		39.5	20.0	10.8	32.9	38.3	24.3	18.9	19.7	17.1					
	NBFNet-QE	Hits@3	31.1	12.6	38.1	22.6	15.1	48.6	69.4	39.6	21.3	12.2	12.8	10.6	21.4	12.4	11.2	7.5
		Hits@10	41.5	21.1	48.1	34.0	24.5	61.8	80.7	52.4	31.1	18.1	22.8	16.4	34.2	22.2	20.1	12.4
217%	NodePiece-QE	Hits@3	5.2		11.1	5.0	4.6	5.0	6.1	4.2	3.8	3.3	3.7					
		Hits@10	11.1		19.6	10.7	9.7	11.7	14.7	10.0	8.4	7.0	8.1					
	NodePiece-QE w/ GNN	Hits@3	18.6		31.3	11.4	6.0	27.7	35.4	17.5	13.3	16.0	9.1					
		Hits@10	31.1		49.1	20.5	12.8	44.6	53.2	30.5	23.2	28.0	17.7					
	NBFNet-QE	Hits@3	37.5	16.9	47.1	26.5	17.4	58.9	75.9	45.4	29.5	21.3	15.1	15.7	29.1	16.2	12.1	11.4
		Hits@10	49.2	27.0	59.9	39.2	28.2	73.9	87.1	59.2	39.5	31.2	25.0	24.2	44.7	27.4	20.0	18.9
175%	NodePiece-QE	Hits@3	5.5		14.8	4.9	4.6	4.7	5.4	3.7	3.9	3.2	3.8					
		Hits@10	11.3		24.1	10.5	10.0	11.0	12.8	9.1	8.6	7.3	8.3					
	NodePiece-QE w/ GNN	Hits@3	25.1		39.6	17.4	9.3	32.1	39.5	22.3	22.0	27.8	15.6					
		Hits@10	37.4		56.5	27.1	16.0	49.3	57.1	35.7	31.8	39.7	23.6					
	NBFNet-QE	Hits@3	38.9	19.6	54.3	27.9	20.5	59.1	72.2	44.7	29.7	24.4	17.8	18.6	30.6	16.7	16.0	16.1
		Hits@10	51.1	31.4	66.1	40.9	31.2	73.0	83.3	58.3	41.3	37.8	27.8	31.1	44.3	28.4	25.2	28.0
150%	NodePiece-QE	Hits@3	4.9		13.4	4.2	3.9	4.4	5.0	3.4	3.6	3.1	3.5					
		Hits@10	10.4		22.5	9.2	8.5	10.3	12.0	8.4	8.0	7.1	7.4					
	NodePiece-QE w/ GNN	Hits@3	22.2		38.0	14.2	6.1	30.3	37.0	19.6	18.6	23.5	12.7					
		Hits@10	35.1		54.5	24.1	11.8	47.4	55.8	32.7	30.1	37.8	22.0					
	NBFNet-QE	Hits@3	39.0	22.5	54.1	26.6	18.5	58.9	71.8	43.9	28.8	28.7	19.2	24.2	32.3	18.0	17.1	21.0
		Hits@10	51.3	35.3	66.5	39.7	29.3	72.8	82.8	57.5	40.6	42.7	29.8	37.6	46.3	29.6	26.8	36.0
133%	NodePiece-QE	Hits@3	6.7		12.8	6.6	3.6	7.0	8.2	5.2	6.7	5.4	4.8					
		Hits@10	13.7		23.5	13.2	7.8	15.0	17.7	11.4	13.8	11.0	10.0					
	NodePiece-QE w/ GNN	Hits@3	24.7		43.6	16.6	6.4	32.9	39.0	22.7	22.6	25.4	12.8					
		Hits@10	37.4		59.5	26.3	11.8	50.1	57.4	35.9	34.3	39.5	21.4					
	NBFNet-QE	Hits@3	40.6	21.1	57.4	29.2	21.3	60.0	70.1	44.0	31.7	30.1	21.6	23.1	25.5	18.3	17.7	20.9
		Hits@10	53.5	34.2	70.7	43.5	31.9	74.2	81.7	57.6	45.0	44.5	32.1	36.9	39.8	30.2	27.9	36.2
121%	NodePiece-QE	Hits@3	6.4		13.9	4.9	3.4	7.5	8.3	4.4	5.5	5.4	4.2					
		Hits@10	12.3		23.8	9.6	7.2	14.1	16.5	9.9	11.0	10.5	8.5					
	NodePiece-QE w/ GNN	Hits@3	26.2		47.2	16.7	7.3	34.4	40.4	21.3	24.0	29.8	14.5					
		Hits@10	38.5		62.4	25.6	12.2	50.3	58.4	34.1	35.5	44.1	23.6					
	NBFNet-QE	Hits@3	38.6	20.6	60.6	25.8	20.5	55.2	65.4	41.4	29.6	29.3	19.6	21.2	28.5	16.3	16.2	20.9
		Hits@10	50.8	31.4	72.7	38.2	32.2	69.0	78.0	54.4	41.2	41.0	30.8	31.9	41.0	27.4	25.4	31.3
113%	NodePiece-QE	Hits@3	5.6		13.8	4.5	3.2	5.4	5.5	3.7	5.2	5.0	3.9					
		Hits@10	11.7		24.8	9.7	7.2	12.9	14.1	8.7	10.3	9.7	8.2					
	NodePiece-QE w/ GNN	Hits@3	28.1		47.3	17.6	7.1	35.4	40.5	24.3	28.5	35.2	16.4					
		Hits@10	40.2		62.7	26.7	11.7	51.4	57.1	36.6	41.0	49.7	24.6					
	NBFNet-QE	Hits@3	34.8	18.2	58.2	21.1	16.4	52.1	62.6	36.2	23.7	27.0	15.5	20.6	21.8	13.9	14.4	20.5
		Hits@10	47.5	28.6	71.2	33.8	28.0	67.8	75.5	49.0	35.9	38.6	28.2	31.2	33.2	24.0	23.6	30.8
106%	NodePiece-QE	Hits@3	5.0		11.2	4.6	3.3	4.8	4.6	3.6	5.1	4.2	3.9					
		Hits@10	10.7		23.0	9.0	6.8	11.8	11.7	8.1	10.2	8.1	7.4					
	NodePiece-QE w/ GNN	Hits@3	30.7		47.1	22.5	9.7	33.2	38.2	26.5	36.0	43.2	20.2					
		Hits@10	41.1		63.4	30.3	13.8	47.3	54.3	37.5	44.2	53.3	26.2					
	NBFNet-QE	Hits@3	33.7	19.0	60.0	21.0	16.7	48.8	53.0	32.8	25.0	29.2	16.7	21.6	18.3	12.5	14.8	27.8
		Hits@10	45.2	27.6	72.9	33.1	26.9	63.1	65.3	44.0	36.6	37.0	28.0	29.8	28.4	22.1	24.2	33.5

Fig. 6 and Table 8 contain detailed results from the experiment in Section 5.3 about executing **training** queries over the original *training* and extended *test inference* graphs.

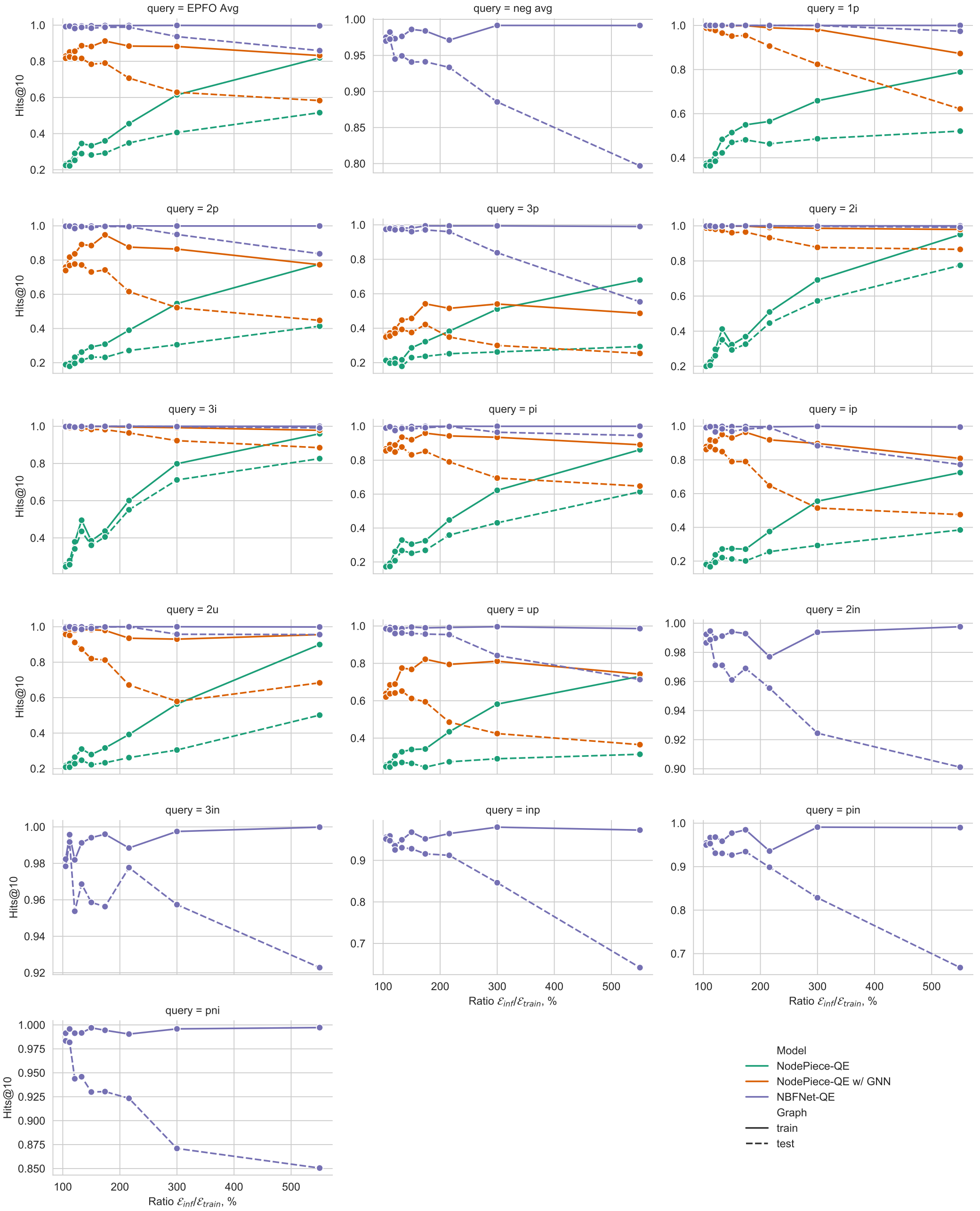

Figure 6: Hits@10 results on answering **training** queries executed over the original train (solid line) and test inference (dashed line) graphs. NodePiece-QE models are inference-only and were trained on *1p* queries, NBFNet-QE is end-to-end trainable on all complex queries.

Table 8: Hits@10 results (%) of **training** queries executed over the original *training* graph and extended *test inference* graph. All ratios $\mathcal{E}_{inf}/\mathcal{E}_{train}$. avg$_p$ is the average on EPFO queries ($\wedge$, $\vee$). avg$_n$ is the average on queries with negation. NodePiece-QE models are inference-only and were trained on *1p* queries, NBFNet-QE is end-to-end trainable on all complex queries.

Ratio	Model	Graph	**avg**$_p$	**avg**$_n$	1p	2p	3p	2i	3i	pi	ip	2u	up	2in	3in	inp	pin	pni
550%	NodePiece-QE	train	81.9		78.9	77.7	68.0	95.0	96.0	86.2	72.5	89.9	73.1					
		test	51.6		52.1	41.4	29.4	77.5	82.6	61.4	38.5	50.1	31.3					
	NodePiece-QE w/ GNN	train	83.2		87.3	77.3	48.6	97.9	97.8	89.1	80.9	95.5	74.2					
		test	58.3		62.1	44.7	25.4	86.6	88.5	64.7	47.6	68.4	36.5					
	NBFNet-QE	train	99.7	99.1	100.0	99.9	99.0	100.0	100.0	100.0	99.5	99.9	98.6	99.8	100.0	97.3	99.0	99.7
		test	85.9	79.7	97.4	83.7	55.3	99.1	99.1	94.5	77.2	95.6	71.3	90.1	92.3	64.2	66.8	85.1
300%	NodePiece-QE	train	61.4		65.9	54.6	51.1	69.2	79.9	62.2	55.5	56.4	58.2					
		test	40.6		48.6	30.5	26.2	57.2	71.2	43.1	29.3	30.5	28.9					
	NodePiece-QE w/ GNN	train	88.2		98.2	86.4	54.1	98.6	99.3	93.5	89.7	93.0	81.1					
		test	62.9		82.4	52.2	30.0	87.7	92.3	69.5	51.5	57.9	42.4					
	NBFNet-QE	train	99.9	99.2	100.0	99.9	99.4	100.0	100.0	100.0	99.9	100.0	99.6	99.4	99.7	98.0	99.1	99.6
		test	93.7	88.5	99.9	95.0	83.8	99.8	99.8	96.5	88.4	95.8	84.2	92.4	95.7	84.6	82.8	87.1
217%	NodePiece-QE	train	45.5		56.5	39.0	38.3	50.9	60.1	44.7	37.5	39.2	43.4					
		test	34.8		46.3	27.1	25.2	44.6	55.1	35.9	25.6	26.2	27.3					
	NodePiece-QE w/ GNN	train	88.4		98.9	87.6	51.5	99.1	99.5	94.3	91.9	93.5	79.4					
		test	70.7		90.6	61.6	34.9	93.3	96.4	79.0	64.7	67.1	48.5					
	NBFNet-QE	train	99.8	97.1	100.0	99.9	99.4	100.0	100.0	100.0	99.6	100.0	99.2	97.7	98.8	96.4	93.6	99.0
		test	98.9	93.3	100.0	99.5	96.0	100.0	100.0	99.9	99.2	100.0	95.4	95.5	97.8	91.2	89.9	92.3
175%	NodePiece-QE	train	36.0		54.9	30.8	32.2	36.9	43.7	32.5	27.1	31.6	34.2					
		test	29.2		48.1	23.2	23.7	32.6	40.5	26.9	20.1	23.3	24.4					
	NodePiece-QE w/ GNN	train	91.2		99.9	94.7	54.2	99.9	99.9	96.0	96.4	97.9	82.2					
		test	79.0		95.4	74.2	42.2	96.5	98.2	85.2	79.0	81.2	59.4					
	NBFNet-QE	train	99.8	98.4	100.0	99.9	99.4	100.0	100.0	99.9	99.8	100.0	99.0	99.3	99.6	95.2	98.5	99.4
		test	98.9	94.1	100.0	99.7	97.1	100.0	100.0	99.2	98.3	99.9	95.7	96.9	95.6	91.6	93.5	93.0
150%	NodePiece-QE	train	33.3		51.4	29.2	28.6	32.3	38.5	30.5	27.4	27.9	33.9					
		test	28.2		47.0	23.3	23.0	29.3	36.0	25.2	21.2	22.2	26.5					
	NodePiece-QE w/ GNN	train	88.2		99.8	88.5	45.7	99.8	99.9	92.1	93.0	98.3	76.8					
		test	78.4		95.2	73.1	37.5	96.1	98.3	83.2	79.0	82.0	61.2					
	NBFNet-QE	train	99.7	98.6	100.0	99.8	98.1	100.0	100.0	99.8	99.7	100.0	99.4	99.4	99.4	96.8	97.7	99.7
		test	98.3	94.1	100.0	98.8	96.1	99.9	99.9	98.4	96.9	99.0	96.0	96.1	95.9	92.8	92.7	93.0
133%	NodePiece-QE	train	34.5		48.4	26.2	21.7	41.2	49.5	32.9	27.2	31.0	32.7					
		test	28.9		42.2	21.3	17.9	35.1	43.5	26.8	22.1	24.7	27.0					
	NodePiece-QE w/ GNN	train	88.7		99.8	89.1	44.7	99.9	99.9	93.6	95.0	98.3	77.5					
		test	81.6		96.5	77.2	39.3	97.4	98.8	87.8	84.9	87.3	65.2					
	NBFNet-QE	train	99.5	97.6	100.0	99.9	98.2	100.0	99.9	99.4	99.8	99.9	98.6	99.1	99.1	94.9	95.9	99.2
		test	98.7	94.9	100.0	99.6	97.4	99.9	99.9	98.7	98.4	98.5	96.3	97.1	96.9	93.0	93.1	94.6
121%	NodePiece-QE	train	29.1		41.9	23.2	22.2	29.7	37.9	26.2	23.7	26.4	30.6					
		test	25.2		38.5	19.6	19.7	26.0	34.1	20.8	19.2	22.8	26.3					
	NodePiece-QE w/ GNN	train	85.6		99.9	83.6	39.6	99.8	99.9	88.9	91.1	98.1	68.9					
		test	81.8		97.7	77.7	37.1	98.1	99.2	84.8	86.1	91.2	64.1					
	NBFNet-QE	train	99.4	97.3	100.0	99.9	98.0	99.8	99.8	98.7	99.6	99.9	99.0	99.0	98.2	93.5	96.8	99.1
		test	98.2	94.5	100.0	98.4	97.1	99.5	99.5	97.5	96.6	98.9	96.0	97.1	95.4	92.5	93.1	94.4
113%	NodePiece-QE	train	24.0		38.2	19.6	20.9	22.5	27.8	19.2	18.1	22.8	26.5					
		test	22.1		36.3	17.9	19.7	20.5	25.6	17.4	16.6	20.8	24.5					
	NodePiece-QE w/ GNN	train	85.2		99.8	81.7	37.3	99.7	99.9	89.2	91.8	98.6	68.5					
		test	82.5		98.5	76.9	35.5	98.6	99.4	86.6	87.8	95.1	63.8					
	NBFNet-QE	train	99.7	98.2	100.0	100.0	98.3	100.0	100.0	99.6	99.9	100.0	99.2	99.5	99.6	95.9	96.7	99.6
		test	99.5	97.3	100.0	99.8	97.9	100.0	100.0	99.7	99.7	100.0	98.0	98.9	99.2	94.7	95.3	98.2
106%	NodePiece-QE	train	23.1		37.4	19.4	21.8	20.5	25.4	17.9	18.7	21.7	25.5					
		test	22.4		36.5	18.9	21.3	20.0	24.5	17.2	18.0	20.8	24.7					
	NodePiece-QE w/ GNN	train	82.9		99.6	75.9	35.6	99.4	99.7	86.7	87.9	97.3	63.9					
		test	81.7		98.8	73.9	34.9	98.8	99.4	85.5	86.2	95.7	62.0					
	NBFNet-QE	train	99.5	97.5	100.0	99.9	98.0	99.9	99.6	99.1	99.7	99.9	99.1	99.2	98.2	95.5	95.4	99.1
		test	99.2	97.0	100.0	99.8	97.4	99.8	99.8	99.0	99.1	99.2	98.5	98.7	97.8	95.1	95.0	98.3

D Hyperparameters

Both NodePiece-QE and NBFNet-QE models are implemented with PyTorch [21] (MIT License). In particular, NodePiece-QE models employ PyG [11] (MIT License) and PyKEEN [2] (MIT License) for training link prediction models. NBFNet-QE is implemented based on the official NBFNet repository [6] (MIT License) and TorchDrug [38] library (Apache 2.0).

For all inductive experiments in Sections 5.2 and 5.3, Table 9 lists best hyperparameters for NodePiece-QE models without GNN encoder, Table 10 contains hyperparameters for GNN-enabled NodePiece-QE models. The GNN-free models use only relation-based tokenization where each entity e is represented with two fixed-size sets: a set of k unique *incoming* r_i and a set of k unique *outgoing* r_o relation types. Looking up their d-dimensional vectors, we obtain:

$$e = \Big[[\boldsymbol{r}_{i1}, \boldsymbol{r}_{i2}, \dots, \boldsymbol{r}_{ik}][\boldsymbol{r}_{o1}, \boldsymbol{r}_{o2}, \dots, \boldsymbol{r}_{ok}]\Big] \in \mathbb{R}^{2\times k\times d}$$

If, for some entity, the number of unique relations of a certain kind is less than k, we pad the set with auxiliary `[PAD]` tokens. Entity representations are built as a function of the two sets $f(e) : \mathbb{R}^{2\times k\times d} \to \mathbb{R}^d$:

$$\boldsymbol{h}_e = \text{MLP}\Big(\text{RP}(\sum_{j=0}^{k} \boldsymbol{r}_{ij}) + \text{RP}(\sum_{j=0}^{k} \boldsymbol{r}_{oj})\Big)$$

Particularly, we first sum up tokens of the same direction, pass them through a random projection layer RP (we found that making this projection learnable does not improve results), sum up representations of *incoming* and *outgoing* parts, and pass the resulting vector through a learnable MLP. This way, the number of learnable encoder parameters does not depend on the sequence length k, i.e., the number of chosen tokens per node.

The GNN-enabled models employ a slightly different *Concat + MLP* encoder where each node is tokenized with a sample of k incident relations. Then, we concatenate d-dimensional embeddings of those *tokens* t_i into a single long vector $\mathbb{R}^{kd}$, and then use a 2-layer MLP to project it to a model dimension d, i.e., $f(e) : \mathbb{R}^{kd} \to \mathbb{R}^d$:

$$\boldsymbol{h}_e = \text{MLP}\Big([\boldsymbol{t}_0; \boldsymbol{t}_1; \dots; \boldsymbol{t}_k]\Big)$$

For the large-scale experiment on WikiKG-QE in Section 5.4, we employ the *Concat + MLP* encoder. Instead of separating incoming and outgoing relation types, we first tokenize each node with 20 nearest *anchors* (pre-selected in advance using the default NodePiece strategy [12]) and add a sample of k unique incident relations.

The overall tokens vocabulary consists of 20,000 anchor nodes, 1,024 relation types (including inverse relations) and one `[PAD]` token. All hyperparameters for this experiments are listed in Table 11.

Having trained the link predictors, we tune CQD-Beam hyperparameters on the validation set varying the t-norms, t-conorms, and scores normalization. Table 12 lists best options for each EPFO query type. For all experiments, we used a beam size $k = 32$ except for queries on WikiKG-QE where we used $k = 8$ due to the memory-expensive need of maintaining a beam over 2M entities.

Table 13 lists hyperparameters for NBFNet-QE models for all inductive splits. We found this architecture is quite stable under various configurations and eventually employed the same set of hyperparameters across all datasets.

The experiment on the tranductive baseline BetaE (Section 5.2) assumes random initialization of new nodes at inference time and confirms that pure transductive models can not generalize to graphs with unseen nodes. BetaE was configured with $400d$ embedding dimension, batch size 512, 32 negative samples, learning rate 0.0005, margin γ 60, and trained on 10 query patterns *{1p,2p,3p,2i,3i,2in,3in,inp,pin,pni}* for $200k$ steps.

[6] `https://github.com/DeepGraphLearning/NBFNet`

Table 9: NodePiece-QE hyperparameters for all inductive splits.

Hyperparameter	**Dataset $\mathcal{E}_{inf}/\mathcal{E}_{train}$ Ratios**								
	106	**113**	**133**	**134**	**150**	**175**	**217**	**300**	**550**
Vocab size	472	466	460	458	450	438	442	402	346
Tokens per node	50	50	50	50	50	50	30	30	30
Vocab dim	400	400	400	400	400	400	400	400	1000
Scoring function	ComplEx [29]								
Encoder	MLP								
Encoder dim	400	400	400	400	400	400	400	400	1000
Encoder layers	2								
Batch size	256								
Epochs	19	19	28	100	30	70	100	150	510
Learning rate	1e-4								
Optimizer	Adam								
Loss function	Negative Sampling Self-adversarial Loss [26]								
Margin	3	3	3	10	3	10	3	3	3
# negatives	30	30	30	30	30	30	50	50	20
# parameters	510k	508k	505k	504k	501k	496k	498k	482k	2.35M
Training time, h	0.75	0.6	0.7	2	0.5	0.75	0.6	0.6	2.3

E Identifying Easy and Hard Answers

In addition to evaluating *faithfullness* that measures whether a model could recover easy answers, it is also interesting to measure whether all easy answers can be ranked higher than hard answers. That is, a reliable query answering model would first recover all possible easy answers and would enrich the answer set with highly-probable hard answers.

To this end, we apply a ROC AUC metric over original **unfiltered** scores.

The idea of computing ROC AUC is as follows: Suppose we have a list of unfiltered raw scores from which we extract scores of all easy and hard answers. Suppose we have a query with 4 easy and 1 hard answer: $[5, 6, 7, 8, 32]$ where 8 is a rank of a hard answer while 5, 6, 7, 32 are ranks of easy answers. We then create binary labels for the scores assigning 1 to the hard answers, e.g., $[0, 0, 0, 1, 0]$.

Given those two arrays, we then compute the ROC AUC score that would measure how many hard answers are ranked after easy answers, e.g., in our example ROC AUC is 0.75. Note that the score does not depend on actual values of ranks, that is, the metric will be high when easy answers are, e.g., ranked 1000-1004 as long as hard answers are ranked 1005 and lower. Therefore, ROC AUC still needs to be paired with MRR to see where easy and hard answers are ranked absolutely.

We compute ROC AUC for each query and average them over each query type thus making it **macro-averaged ROC AUC**. Our experimental results on all query types using the models reported in Table 1 on the reference 175% dataset are compiled in Table 14.

NBFNet-QE performs almost perfectly w.r.t. ROC AUC as it was trained on complex queries. NodePiece-QE models are acceptable for inference-only models that were only trained only on 1p simple link prediction and have never seen any complex query at training time.

Table 10: NodePiece-QE with GNN hyperparameters for all inductive splits.

Hyperparameter	**Dataset $\mathcal{E}_{inf}/\mathcal{E}_{train}$ Ratios**								
	106	**113**	**133**	**134**	**150**	**175**	**217**	**300**	**550**
Vocab size	472	466	460	458	450	438	442	402	346
Tokens per node	50								
Vocab dim	200								
Scoring function	ComplEx [29]								
Encoder	MLP								
Encoder dim	200								
Encoder layers	2								
GNN encoder	CompGCN [30] + attention aggregator + RotatE [26] message function								
GNN layers	3								
GNN dim	200								
GNN attn dropout	0.1								
GNN attn heads	2								
Batch size	256								
Epochs	100	100	250	310	400	500	500	1000	510
Learning rate	1e-4								
Optimizer	Adam								
Loss function	Negative Sampling Self-adversarial Loss [26]								
Margin	20	20	20	20	20	20	20	20	5
# negatives	20	20	30	20	30	30	30	30	50
# parameters	539k	537k	536k	536k	534k	532k	533k	525k	513k
Training time, h	30	23	34	25	17	12	5	5	2

Table 11: NodePiece-QE hyperparameters for WikiKG-QE.

Hyperparameter	**WikiKG-QE (133%)**
Vocab size	20,000 anchors + 1024 relation types
Tokens per node	20 nearest anchors + 20 relations
Vocab dim	100
Scoring function	ComplEx [29]
Encoder	Concat + MLP
Encoder dim	200
Encoder layers	2
Batch size	512
Epochs	40 ($\approx$ 1M steps)
Learning rate	1e-4
Optimizer	Adam
Loss function	BCE
Adversarial temp.	1.0
# negatives	64
# parameters	2,922,900
Training time, h	40

Table 12: CQD-Beam t-norm hyperparameters for all splits and both link predictors, NodePiece-QE and NodePiece-QE w/ GNN, when answering EPFO queries. The default beam size $k = 32$, *prod* + σ is product t-norm with sigmoid score normalization. Details on t-norms are in Appendix A.

Ratio	Link predictor	1p	2p	3p	2i	3i	pi	ip	2u	up
106%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	min prod + σ
113%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	min prod + σ
122%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	min prod + σ
134%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	min prod + σ	min prod + σ	prod + σ	prod + σ	min prod + σ	min prod + σ	prod + σ	prod + σ
150%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ
175%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ
217%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ
300%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ
550%	NodePiece-QE NodePiece-QE w/ GNN	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ	prod + σ
133%	NodePiece-QE	(WikiKG-QE) prod + σ for all query types								

Table 13: Hyperparameters of NBFNet-QE on different datasets. All the hyperparameters are selected by the performance on the validation set.

Hyperparameter		**All splits**
GNN	#layers	4
	hidden dim.	32
	composition	DistMult [32]
	aggregation	PNA [9]
MLP	#layer	2
	hidden dim.	64
Traversal Dropout	probability	0.5
Logical Operator	t-norm	product
Learning	batch size	64
	sample weight	uniform across queries
	loss	BCE
	# negatives	32
	optimizer	Adam
	learning rate	5e-3
	iterations (#batch)	10,000
	adv. temperature	0.1

Table 14: Macro-averaged ROC AUC score over **unfiltered** predictions to measure if all easy answers are ranked higher than hard answers. Higher is better.

Model	**avg**$_p$	**avg**$_n$	**1p**	**2p**	**3p**	**2i**	**3i**	**pi**	**ip**	**2u**	**up**	**2in**	**3in**	**inp**	**pin**	**pni**
Inductive Inference-only																
NodePiece-QE	0.653		0.609	0.656	0.666	0.628	0.627	0.645	0.681	0.677	0.686					
NodePiece-QE w/ GNN	0.675		0.688	0.689	0.667	0.655	0.624	0.646	0.690	0.713	0.701					
Inductive Trainable																
NBFNet-QE	0.978	0.901	0.997	0.981	0.962	0.987	0.978	0.975	0.975	0.982	0.965	0.919	0.884	0.888	0.913	0.904