Neural Graph Reasoning: Complex Logical Query Answering Meets Graph Databases

Anonymous authors
Paper under double-blind review

Abstract

Complex logical query answering (CLQA) is a recently emerged task of graph machine learning that goes beyond simple one-hop link prediction and solves a far more complex task of multi-hop logical reasoning over massive, potentially incomplete graphs in a latent space. The task received a significant traction in the community; numerous works expanded the field along theoretical and practical axes to tackle different types of complex queries and graph modalities with efficient systems. In this paper, we provide a holistic survey of CLQA with a detailed taxonomy studying the field from multiple angles, including graph types (modality, reasoning domain, background semantics), modeling aspects (encoder, processor, decoder), supported queries (operators, patterns, projected variables), datasets, evaluation metrics, and applications.

Refining the CLQA task, we introduce the concept of *Neural Graph Databases* (NGDBs). Extending the idea of graph databases (graph DBs), NGDB consists of a *Neural Graph Storage* and a *Neural Graph Engine*. Inside Neural Graph Storage, we design a graph store, a feature store, and further embed information in a latent embedding store using an encoder. Given a query, Neural Query Engine learns how to perform query planning and execution in order to efficiently retrieve the correct results by interacting with the Neural Graph Storage. Compared with traditional graph DBs, NGDBs allow for a flexible and unified modeling of features in diverse modalities using the embedding store. Moreover, when the graph is incomplete, they can provide robust retrieval of answers which a normal graph DB cannot recover. Finally, we point out promising directions, unsolved problems and applications of NGDB for future research.

Contents

1 Introduction **2**

2 Preliminaries **4**

2.1 Types of Graphs 4

2.2 Basic Graph Query Answering 7

2.3 Basic Approximate Graph Query Answering 8

2.4 Graph Query Answering 8

2.5 Approximate Graph Query Answering 11

2.6 Triangular Norms and Conorms 12

2.7 Graph Representation Learning 12

3 Neural Graph Databases **13**

3.1 Neural Graph Storage . . . 16
3.2 Neural Query Engine . . . 17

4 Graphs **21**
4.1 Modality . . . 21
4.2 Reasoning Domain . . . 21
4.3 Background Semantics . . . 22

5 Modeling **25**
5.1 Encoder . . . 26
5.2 Processor . . . 28
5.3 Decoder . . . 35
5.4 Computation Complexity . . . 36

6 Queries **37**
6.1 Query Operators . . . 37
6.2 Query Patterns . . . 43
6.3 Projected Variables . . . 45

7 Datasets and Metrics **47**
7.1 Evaluation Setup . . . 47
7.2 Query Types . . . 47
7.3 Training . . . 48
7.4 Inference . . . 49
7.5 Metrics . . . 51

8 Applications **52**

9 Summary and Future Opportunities **53**

1 Introduction

Graph databases (graph DBs) are key architectures to capture, organize and navigate structured relational information over real-world entities. Unlike traditional relational DBs storing information in tables with a rigid schema, graph DBs store information in the form of heterogeneous graphs, where nodes represent entities and edges represent relationships between entities. In graph DBs, a relation (i.e. heterogeneous connection between entities) is the first-class citizen. With the graph structure and a more flexible schema, graph DBs allow for a more efficient and expressive way to handle higher-order relationships between distant entities, especially navigating through multi-hop hierarchies. While traditional DBs require expensive join operations to retrieve information, graph DBs can directly traverse the graph and navigate through links more efficiently with the adjacency matrix. Due to its capabilities, graph databases serve as the backbone of many critical industrial applications including question answering in virtual assistants (Flint, 2021; Ilyas et al., 2022), recommender systems in marketplaces (Dong, 2018; Hamad et al., 2018), social networking

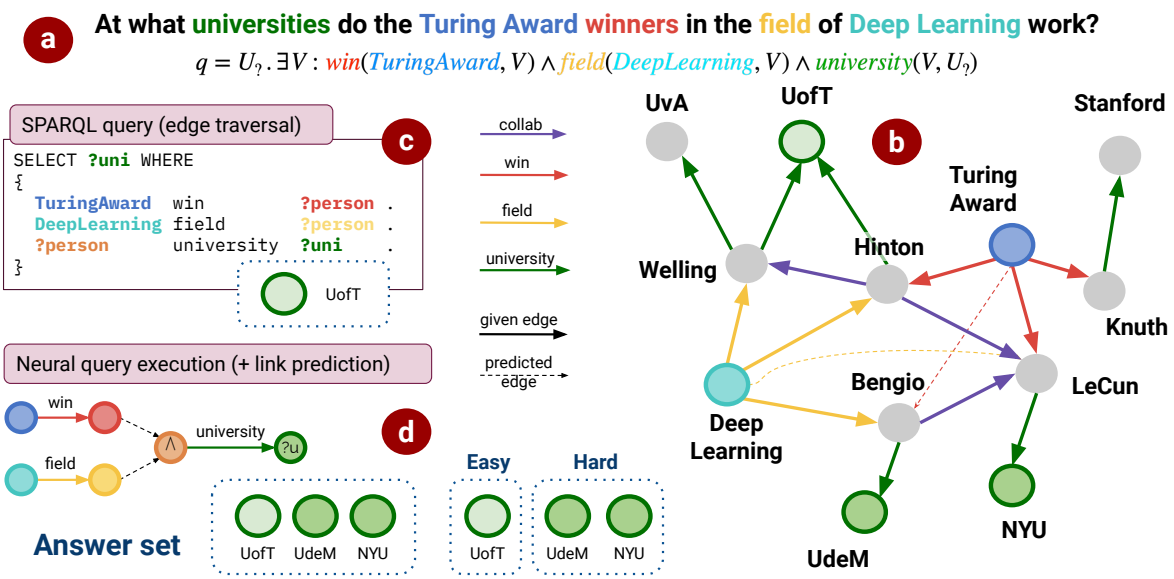

Figure 1: A complex logical query (a) and its execution over an incomplete graph (b). Symbolic engines like SPARQL (c) perform edge traversal and retrieve an incomplete set of *easy* answers directly reachable in the graph, i.e., {`UofT`}. Neural query execution (d) recovers missing ground truth edges (dashed) and returns an additional set of *hard* answers {`UdeM`, `NYU`} unattainable by symbolic methods.

in mobile applications (Bronson et al., 2013), and fraud detection in financial industries (Tian et al., 2019; Pourhabibi et al., 2020).

Given a downstream task, one of the most important tasks of graph DBs is to perform complex query answering. The goal is to retrieve the answers of a given input query of interest from the graph database. Given the query, graph DBs first translate and optimize the query into a more efficient graph traversal pattern with a query planner, and then execute the pattern on the graph database to retrieve the answers from the graph storage using the query executor. The storage compresses the graphs into symbolic indexes suitable for fast table lookups. Querying is thus fast and efficient under the assumption of completeness, i.e., stored graphs have no missing edges.

However, most real-world graphs are notoriously incomplete, e.g., in Freebase, 93.8% of people have no place of birth and 78.5% have no nationality (Mintz et al., 2009), about 68% of people do not have any profession (West et al., 2014), while in Wikidata, about 50% of artists have no date of birth (Zhang et al., 2022a), and only 0.4% of known buildings have information about height (Ho et al., 2022). Naïvely traversing the graph in light of incompleteness leads to a significant miss of relevant results and the issue further exacerbates with an increasing number of hops. This inherently hinders the application of graph databases. Link prediction is a challenging task, prior works predict links by learning a latent representation of each link (Bordes et al., 2013; Yang et al., 2015; Trouillon et al., 2016; Sun et al., 2019) or mining rules (Galárraga et al., 2013; Xiong et al., 2017; Lin et al., 2018; Qu et al., 2021). However, it is always a trade-off between possibly incomplete results and decidability – with a denser graph, some SPARQL entailment regimes (Hawke et al., 2013) do not guarantee that query execution terminates in finite time.

On the other hand, recent advances in graph machine learning enabled expressive reasoning over large graphs in a latent space without facing decidability bottlenecks. The seminal work of Hamilton et al. (2018) on Graph Query Embedding (GQE) laid foundations of answering complex, database-like logical queries over incomplete KGs where inferring missing links during query execution is achieved via parameterization of entities, relations, and logical operators with learnable vector representations and neural networks. For the incomplete knowledge graph in (Fig. 1), given a complex query *"At what universities do the Turing Award winners in the field of Deep Learning work?"*, traditional symbolic graph DBs (SPARQL- or Cypher-

like) would return only one answer (`UofT`), reachable by edge traversal. In contrast, neural query embedding parameterizes the graph and the query with learnable vectors in the embedding space. Neural query execution is akin to *traversing the graph and executing logical operators in the embedding space* that infers missing links and enriches the answer set with two more relevant answers `UdeM` and `NYU` unattainable by symbolic DBs.

Since then, the area has seen a surge of interest with numerous improvements of supported logical operators, query types, graph modalities, and modeling approaches. In our view, those improvements have been rather scattered, without an overall aim. There still lacks a unifying framework to organize the existing works and guide future research. To this end, we propose to present one of the first holistic studies about the field. We devise a taxonomy classifying existing works along three main axes, *i.e.*, (i) **Graphs** (Section 4 – logical formalisms behind the underlying graph and its schema) (ii) **Modeling** (Section 5 – what are the neural approaches to answer queries) (iii) **Queries** (Section 6 – what queries can be answered). We then discuss **Datasets and Metrics** (Section 7 – how we measure performance of NGDB engines). Each of these dimensions is further divided into fine-grained aspects. Finally, we list CLQA applications (Section 8) and summarize open challenges for future research (Section 9).

We further propose a novel framework *Neural Graph Databases* (NGDB, Section 3), where complex approximate query answering (Section 2) is the core task and approached by the neural query embedding methods. NGDB consists of a neural graph storage and a neural query engine, which correspond to a neural version of both counterparts in a regular graph DB respectively. Besides a graph store and feature store that store the graph structure as well as the multimodal node-/edge-level features, neural graph storage assumes an encoder that further compresses the raw information in a semantic-preserving latent embedding space, *i.e.*, similar entities/relations are mapped to similar embeddings. During retrieval, such a design choice instantly allows for both exact identity match as well as approximate nearest neighbor search. As another key component, the neural query engine is responsible for optimizing, planning and executing a given input query in the embedding space, and finally returning the query answers by interacting with the neural graph storage. We provide a detailed conceptual scheme for an NGDB with various levels of design principles and assumptions over the two components. We hope this sheds light on the current state of the art and provides a roadmap for future research on NGDBs.

Related Work. While there exist insightful surveys on general graph machine learning (Chami et al., 2022), simple link prediction in KGs (Ali et al., 2021; Chen et al., 2023), and logic-based link prediction (Zhang et al., 2022b; Delong et al., 2023), the complex query answering area remained uncovered so far. With our work, we close this gap and provide a holistic view on the state of affairs in this emerging field.

2 Preliminaries

Here, we discuss the foundational terms and preliminaries often seen in the database and graph learning literature before introducing neural graph databases. This includes definitions of different types of graphs, graph queries and their mapping to structured languages like SPARQL, formalizing approximate graph query answering, logical operators, and fuzzy logic. The full hierarchy of definitions is presented in Fig. 3.

2.1 Types of Graphs

Here we introduce different types of graphs relevant for this research area. These serve as the core data structure of various query reasoning works and also our neural graph storage module (Section 3.1). Our definitions are adaptations from those in (Hogan et al., 2021, ch. 2).

We begin by defining a set with all elements used in our graph

Definition 2.1 (`Con`) *`Con` is an infinite set of constants.*[1]

Definition 2.2 ((Standard / Triple based) Knowledge Graph) *We define a knowledge graph (KG) $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$, where $\mathcal{E} \subset$ `Con` represents a set of nodes (entities), $\mathcal{R} \subset$ `Con` is a set of relations (edge types),*

[1]In case the constants do not include representations for the real numbers, this set would be countably infinite.

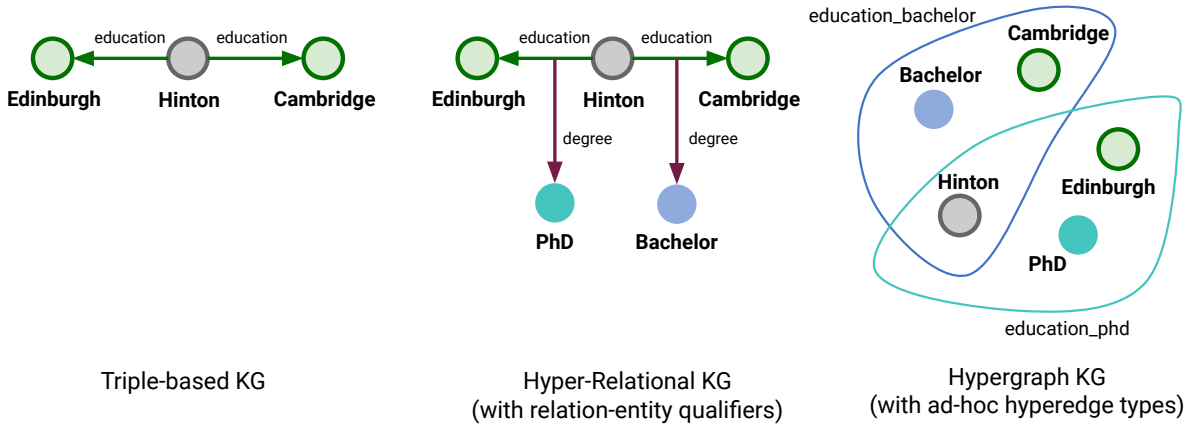

Figure 2: An example of KG modalities. Triple-only graphs have directed binary edges between nodes. Hyper-relational graphs allow key-value relation-entity attributes over edges. Hypergraphs consist of hyper-edges composed of multiple nodes.

and $\mathcal{S} \subset (\mathcal{E} \times \mathcal{R} \times \mathcal{E})$ is a set of edges (triples) [2]*. Each edge $s \in \mathcal{S}$ on a KG $\mathcal{G}$ denotes a statement (or fact) (e_s, r, e_o) in a triple format or $r(e_s, e_o)$ as a formula in first-order logic (FOL) form, where $e_s, e_o \in \mathcal{E}$ denote the subject and object entities, and $r \in \mathcal{R}$ denotes the relationship between the subject and the object. Often, the graph is restricted such that the set of nodes and edges must be finite, we will make this assumption unless indicated otherwise.*

For example, one statement on the KG in Fig. 1b is `(Hinton, university, UofT)` or `university(Hinton, UofT)`. Note all relations are binary on KGs, *i.e.*, each relation involves exactly two entities. This is also the choice made for the resource description framework (RDF) (Brickley et al., 2014) that defines a triple as a basic representation of a fact, yet RDF allows a broader definition of triples, where the object can be literals including numbers and timestamps (detailed in the following paragraphs).

Hyper-relational KGs generalize triple KGs by allowing edges to have relation-entity qualifiers. We define them as follows:

Definition 2.3 (Hyper-relational Knowledge Graph - derived from Alivanistos et al. (2022)) *With $\mathcal{E}$ and $\mathcal{R}$ defined as in Definition 2.2, let $\mathfrak{Q} = 2^{(\mathcal{R} \times \mathcal{E})}$, we define a hyper-relational knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$. In this case, the edge set $\mathcal{S} \subset (\mathcal{E} \times \mathcal{R} \times \mathcal{E} \times \mathfrak{Q})$ consists of qualified statements, where each statement $s = (e_s, r, e_o, qp)$ includes qp that represent contextual information for the main relation r. This set $qp = \{q_1, \ldots\} = \{(qr_1, qe_1), (qr_2, qe_2) \ldots\} \subset \mathcal{R} \times \mathcal{E}$ is the set of* qualifier pairs*, where $\{qr_1, qr_2, \ldots\}$ are the* qualifier relations *and $\{qe_1, qe_2, \ldots\}$ the* qualifier entities*.*

We note that if $\mathcal{E}$ and $\mathcal{R}$ are finite sets, then also $\mathfrak{Q}$ is finite and, there are only a finite number of (r, qp) combinations possible. As a consequence, we find that with these conditions, and by defining a canonical ordering over $\mathfrak{Q}$, we can represent the hyper-relational graph using first order logic by coining a new predicate r_{qp} for each combination. The statement from the definition can then be written as $r_{qp}(e_s, e_o)$.

For example, one statement on a hyper-relational KG in (Fig. 2) is `(Hinton, education, Cambridge, {(degree, Bachelor)})`. The qualifier pair `{(degree, Bachelor)}` provides additional context to the main triple and helps to distinguish it from other `education` facts. If the conditions mentioned above are met, we can write this statement as $\texttt{education}_{\{(\texttt{degree:Bachelor})\}}(\texttt{Hinton}, \texttt{Cambridge})$.

Hyper-relational KGs can capture a more diverse set of facts by using the additional qualifiers for facts on the graph. This concept is closely connected to the RDF-star (previously known as RDF*) format introduced in Hartig et al. (2022), as an extension of RDF. RDF-star explicitly separates metadata (context) from data

[2]There might be an overlap between $\mathcal{E}$ and $\mathcal{R}$, i.e., some constants might be used as an edge and as a node.

(statements). RDF-star supports a superset of hyper-relational graphs; it extends the values allowed in the object and qualifier value position to include literals, which we will discuss below.

Definition 2.4 (Hypergraph Knowledge Graph) *With $\mathcal{E}$ and $\mathcal{R}$ defined as in Definition 2.2, a hypergraph KG is a graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$ where statements $\mathcal{S} \subset (\mathcal{R} \times 2^{\mathcal{E}})$ are hyperedges. Such a hyperedge $s = (r, e) = (r, \{e_1, \ldots, e_k\})$ has one relation type r and links the k entities together, where the order of these entities is not significant. k, the size of e is called the* arity *of the hyperedge.*

An example hyperedge in Fig. 2 is (`education_degree`, {`Hinton`, `Cambridge`, `Bachelor`}). This is a 3-ary statement (being comprised of three entities). In contrast to hyper-relational KGs, hyperedges consist of only one relation type and cannot naturally incorporate more fine-grained relational compositions. Instead, the type of the hyperedge is rather a merged set of statement relations and every composition leads to a new relation type on hypergraph KGs. It is thus not as compositional as the hyper-relational KG. That is, when describing, for instance, a `major` relation of the `education(Hinton, Cambridge)` fact, a hyper-relational KG can simply use it as a qualifier and retain both relations whereas a hypergraph model has to come up with a new hyperedge type `education_major`. With a growing number of relations, such a strategy of creating new hyperedge relation types might lead to a combinatorial explosion. However, the hypergraph model is more suitable for representing relationships of varying arity, with equal contributions of all entities such as `partnership(companyA, companyB, companyC)`.

Each of the graph types introduced above can be extended to also support literals. This happens by allowing nodes or qualifier values to contain a literal value. Some graphs, like property graphs allow nodes to contain attributes, but this is equivalent with creating extra nodes with these attribute values and adding relations to those. In theory, also the edge type could be a literal, but we exclude these from this work.

Definition 2.5 (KG with Literals) *With $\mathcal{E}$ and $\mathcal{R}$ as for one of the graph types above, a corresponding KG with literals $\mathcal{G}_{\mathcal{L}} = (\mathcal{E}, \mathcal{R}, \mathcal{S}_{\mathcal{L}}, \mathcal{L})$ has an additional set of literals $\mathcal{L} \subset$ `Con` representing numerical, categorical, textual, images, sound waves, or other continuous values ($\mathcal{L}$ is disjoint from $\mathcal{E}$ and $\mathcal{R}$). If we extend standard KGs to RDF graphs, literals can only be used in the objects position, that is $\mathcal{S}_{\mathcal{L}} \subset (\mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \mathcal{L}))$. In RDF-star graphs, literals can be objects or qualifier values, that is, $\mathfrak{Q} = 2^{(\mathcal{R} \times (\mathcal{E} \cup \mathcal{L}))}$ and $\mathcal{S}_{\mathcal{L}} \subset (\mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \mathcal{L}) \times \mathfrak{Q})$.[3] For both of these, we could also define graph types with literals in other positions of the triples, as necessary, or introduce more complex substructures in the elements of the triple (see e.g., Cochez (2012)). In hypergraph KGs, literals can be introduced in the elements of a hyperedge, $\mathcal{S}_{\mathcal{L}} \subset (\mathcal{R} \times 2^{\mathcal{E} \cup \mathcal{L}})$*

If the set of possible statements of the KG with literals has the same finiteness properties as the one without literals, then the properties regarding expressing it using first order logic do not change.

An example triple with a literal object (Fig. 10) is `(Cambridge, established, 1209)`. Literals contain numerical, categorical, discrete timestamps, or text data that cannot be easily discretized in an exhaustive set of possible values, like entities. Similarly to the Web Ontology Language (OWL, Motik et al. (2009)) that distinguishes *object properties* that connect two entities from *datatype properties* that connect an entity and a literal, it is common to use the term *relation* for an edge between two entities, and *attribute* for an edge between an entity and a literal. For example, in `education(Hinton, Cambridge)`, `education` is a relation, whereas `established(Cambridge, 1209)` is an attribute.

The knowledge graph definitions above are not exhaustive. It is possible to create graphs with other properties. Examples include graphs with undirected edges, without edge labels, with time characteristics, with probabilistic or fuzzy relations, etc. Besides, it is also possible to have graphs or edges with combined characteristics. One could, for instance, define a hyperedge with qualifying information. Because of this plethora of options, we decided to limit ourselves to the limited set of options above. In the next section we introduce how to query these graphs.

[3]Both RDF and RDF-star also allow blank nodes used to indicate entities without a specified identity in the subject and object position Brickley et al. (2014) . Besides, they also have support for named graphs (and in some cases for quadruples). We do not support these explicitly in our formalism, but all of these can be modeled using hyper-relational graphs.

2.2 Basic Graph Query Answering

We base our definition of basic graph queries on the one from (Hogan et al., 2021, section 2.2.1 basic graph patterns), but adapt it to our graph formalization.

Definition 2.6 (Term and Var) *Given a knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$ (or equivalent $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S}, \mathcal{L})$), we define the set of variables $\texttt{Var} = \{v_1, v_2, \dots\}$ which take values from $\texttt{Con}$, but is strictly disjoint from it. We call the union of the variables and the constants the terms: $\texttt{Term} = \texttt{Con} \cup \texttt{Var}$*

With these concepts in place a Basic Graph Query is defined as follows:

Definition 2.7 (Basic Graph Query) *A basic graph query is a 4-tuple $\mathcal{Q} = (\mathcal{E}', \mathcal{R}', \mathcal{S}', \overline{\mathcal{S}'})$ (equivalently a 5-tuple $\mathcal{Q} = (\mathcal{E}', \mathcal{R}', \mathcal{S}', \overline{\mathcal{S}'}, \mathcal{L}')$, with literals), with $\mathcal{E}' \subset \texttt{Term}$ a set of node terms, $\mathcal{R}' \subset \texttt{Term}$ a set of relation terms, and $\mathcal{S}', \overline{\mathcal{S}'} \subset \mathcal{E}' \times \mathcal{R}' \times \mathcal{E}'$, two sets of edges (or equivalent to how graph edges are defined with literals). The query looks like two (small) graphs; one formed by the edges in $\mathcal{S}'$, and another by the edges in $\overline{\mathcal{S}'}$. The former set includes edges that must be matched in the graph to obtain answers to the query, while the latter contains edges that **must not** be matched (i.e., atomic negation).*

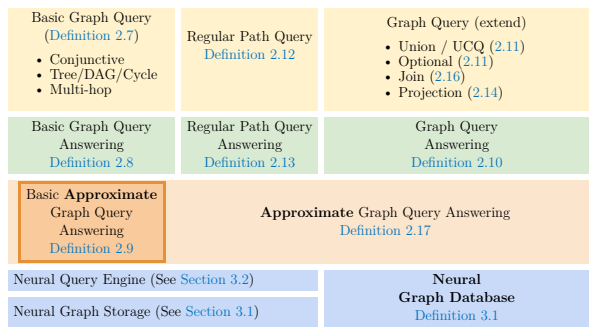

Figure 3: A hierarchy of definitions from Graph Query to Graph Query Answering, a more general Approximate Graph Query Answering leading to Neural Graph Database, NGDB.

With $\texttt{Var}_Q = \mathcal{E}' \cap \texttt{Var}$ (all variables in the query), the answer to the query is defined as follows.

Definition 2.8 (Basic Graph Query Answering) *Given a knowledge graph $\mathcal{G}$ and a query $\mathcal{Q}$ as defined above, an answer to the query is any mapping $\mu : \texttt{Var}_Q \to \texttt{Con}$ such that replacing each variable v_i in the edges in $\mathcal{S}'$ with $\mu(v_i)$ results in an edge in $\mathcal{S}$,* and *replacing each variable v_j in the edges in $\overline{\mathcal{S}'}$ with $\mu(v_j)$ results in an edge that is **not** in $\mathcal{S}$. The set of all answers to the query is the set of all such mappings.*

For triple-based KGs, each edge (a, R, b) of the edge set $\mathcal{S}$ (respectively $\mathcal{S}'$) can be seen as relation projections $R(a, b)$ (resp. $\neg R(a, b)$), *i.e.*, binary functions. Now, because the conjunction (*i.e.*, all) of these relation projections (resp. the negation) have to be true, we also call these *conjunctive* queries with atomic negation (CQ_{neg}). The SPARQL query language defines basic graph patterns (BGPs). These closely resemble our basic graph query for RDF graphs, but with $\overline{\mathcal{S}'} = \emptyset$, *i.e.*, they do not support atomic negation but only conjunctive queries (CQ). We could analogously create CQ and CQ_{neg} classes for the other KG types.

In Fig. 1c we provide an example of a Basic Graph Query expressed in the form of a SPARQL query. This corresponds to our formalism:

$$
\begin{aligned}
\mathcal{Q} = (&\{\texttt{TuringAward}, ?person, \texttt{DeepLearning}, ?uni\}, && (\texttt{\# Node Terms} \quad \mathcal{E}') \\
&\{win, field, university\}, && (\texttt{\# Relation Terms} \quad \mathcal{R}') \\
&\{(\texttt{TuringAward}, win, ?person), (\texttt{DeepLearning}, field, ?person), (?person, university, ?uni)\}, && (\mathcal{S}') \\
&\emptyset) && (\overline{\mathcal{S}'})
\end{aligned}
$$

A possible answer to this query is the following partial mapping: $\mu_1 = \{(?person, \texttt{Hinton}), (?uni, \texttt{UofT})\}$ This is also the only answer, so the set of all answers is $\{\mu_1\}$

If we want to exclude people who have worked together with `Welling`, then we modify the query as follows:

$$
\begin{aligned}
\mathcal{Q} = (&\{\texttt{TuringAward}, ?person, \texttt{DeepLearning}, ?uni\}, \\
&\{win, field, university\}, \\
&\{(\texttt{TuringAward}, win, ?person), (\texttt{DeepLearning}, field, ?person), (?person, university, ?uni)\}, \\
&\mathbf{\{(?person, collab, \texttt{Welling})\}})
\end{aligned}
$$

Note the key difference here is that we add $\{(\textit{?person}, \textit{collab}, \texttt{Welling})\}$ to $\overline{\mathcal{S}'}$. The answer set to this query is empty.

2.3 Basic Approximate Graph Query Answering

Until now, we have assumed that our KG is complete, *i.e.*, it is possible to exactly answer the queries. However, we are interested in the setting where we do not have the complete graph. The situation can be described as follows: Given a knowledge graph $\mathcal{G}$ (subset of a complete, but not observable graph $\hat{\mathcal{G}}$) and a basic graph query $\mathcal{Q}$. Basic Approximate Graph Query Answering is the task of answering $\mathcal{Q}$, without having access to $\hat{\mathcal{G}}$. Depending on the setting, an approach to this could have access to $\mathcal{G}$, or to a set of example (query, answer) pairs, which can be used to produce the answers (see also Section 7.3). In the example from Fig. 1, we also have several edges drawn with dashes. These are edges which are true, but only there in the non-observable part of the graph $\hat{\mathcal{G}}$.

The goal is to find the answer set as if $\hat{\mathcal{G}}$ was known. In this case, the complete answer set becomes $\{\mu_1, \{(\textit{?person}, \texttt{Bengio}), (\textit{?uni}, \texttt{UdeM})\}, \{(\textit{?person}, \texttt{LeCun}), (\textit{?uni}, \texttt{NYU})\}\}$, which includes the answer μ_1 of the non-approximate version.

The query which includes the negation would have the following answers if our graph was complete: $\{\{(\textit{?person}, \texttt{Bengio}), (\textit{?uni}, \texttt{UdeM})\}, \{(\textit{?person}, \texttt{LeCun}), (\textit{?uni}, \texttt{NYU})\}\}$, which does not include μ_1.

Approximate Query Answering Systems provide a score $\in \mathbb{R}$ for *every possible* mapping[4]. Hence, the answer provided by these systems is a function from mappings (*i.e.*, a μ) to their corresponding score in $\mathbb{R}$.

Definition 2.9 (Basic Approximate Graph Query Answering) *Given a knowledge graph $\mathcal{G}$ (subgraph of a complete, but not observable knowledge graph $\hat{\mathcal{G}}$), a basic graph query $\mathcal{Q}$, and the scoring domain $\mathbb{R}$, a* basic approximate graph query answer *to the query $\mathcal{Q}$ is a function f which maps every possible mapping ($\mu : \texttt{Var}_Q \rightarrow \texttt{Con}$) to $\mathbb{R}$.*

The objective is to make it such that the correct answers according to the graph $\hat{\mathcal{G}}$ get a better score (are ranked higher, have a higher probability, have a higher truth value) than those which are not correct answers. However, it is not guaranteed that answers which are in the observable graph $\mathcal{G}$ will get a high score.

For our example, a possible mapping is visualized in Table 1. Each row in the table corresponds to one mapping μ. Ideally, all correct mappings should be ranked on top, and all others below. However, in this example we see several correct answers ranked high, but also two which are wrong.

Table 1: Ordered scored mappings for the example query. The two wrong mappings are in red.

?person	?uni	score
Hinton	UofT	40
Bengio	UdeM	35
Welling	UofT	34
UdeM	Hinton	33
LeCun	NYU	32
...	...	...

2.4 Graph Query Answering

In the previous sections, we introduced the basic graph query and how it can be answered either exactly, or in an approximate graph querying setting. However, there is variation in what types of queries methods can answer; some only support a subset of our basic graph queries, while others support more complicated queries. In this section we will again focus on *exact (non-approximate) query answering* and look at some possible restrictions and then at extensions. We will also highlight links to FOL fragments.

Definition 2.10 (Graph Query Answering) *Given a knowledge graph $\mathcal{G}$, a query formalism, and a graph query $\mathcal{Q}$ according to that formalism. An answer to the query is any mapping $\mu : \texttt{Var}_Q \rightarrow \texttt{Con}$ such that the requirements of the query formalism are met. The set of all answers is the set of all such mappings.*

For our basic graph queries introduced in Definition 2.7, the query formalism sets requirements as to what edges must and must not exist in the graph (Definition 2.8). In that context we already mentioned conjunctive queries, which exist either with (CQ_{neg}) or without (CQ) negation. If the conditions for writing our graphs

[4]This score can indicate the rank of the mapping, a likelihood or a truth value, or be a binary indicating value.

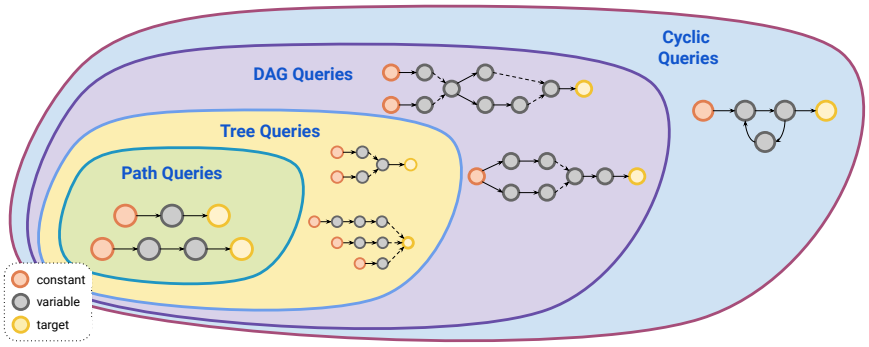

Figure 4: A space of query patterns and their relative expressiveness. Multi-hop reasoning systems tackle the simplest *Path* queries. Existing complex query answering systems support *Tree* queries whereas *DAG* and *Cyclic* queries remain unanswerable by current neural models.

using first order logic (FOL) hold, we can equivalently write our basic graph queries in first order logic. Each variable in the query becomes existentially quantified, and the formula becomes a conjunction of 1) the terms formed from the triples in S', and 2) the negation of the terms formed from the triples in $\overline{S'}$. If there are variables on the edges of the query graph, then we can rewrite the second order formula in first order logic, by interpreting them as a disjunction over all possible predicates from the finite number of options. Our example query from above becomes the following FOL sentence.

$$q = \exists ?person, ?uni : win(\texttt{TuringAward}, ?person) \wedge field(\texttt{DeepLearning}, ?person) \wedge university(?person, ?uni)$$

The answer to the query is the variable assignment function. For several of the more restrictive fragments, it is useful to formulate the queries using this FOL syntax.

Restrictions The first restriction one can introduce are **multi-hop queries**, known in the NLP and KG literature for a long time (Guu et al., 2015; Das et al., 2017; Asai et al., 2020) mostly in the context of question answering. Formally, multi-hop queries (or *path* queries) are CQ which form a chain, where the tail of one projection is a head of the following projection, *i.e.*,

$$q_{\text{path}} = V_k, \exists V_1, \ldots, V_{k-1} : r_1(v, V_1) \wedge r_2(V_1, V_2) \wedge \cdots \wedge r_k(V_{k-1}, V_k)$$

where $v \in \mathcal{E}$, $\forall i \in [1, k] : r_i \in \mathcal{R}, V_i \in \texttt{Var}$ and all V_i are existentially quantified variables. In other words, path queries do not contain branch intersections and can be solved iteratively by fetching the neighbors of the nodes. One could also define multi-hop queries which allow negation.

Other ways of restricting CQ and on CQ_{neg}, resulting in more expressive queries than the multi-hop ones exist. One can define families of logical queries shaped as a *Tree*, a *DAG*, and allowing *cyclic* parts. Illustrated in Fig. 4, path (multi-hop) queries form the least expressive set of logical queries. Tree queries add more reasoning branches connected by intersection operators, DAG queries drop the query tree requirement and allow queries to be directed acyclic graphs, and, finally, cyclic queries further drop the acyclic requirement. Note that these queries do not allow variables in the predicate position. Besides, all entities (in this context referred to as anchors) must occur before all variables in the topological ordering.

We elaborate more on these query types in Section 6.2 and note that the majority of surveyed neural CLQA methods in Section 5 are still limited to Tree-like queries, falling behind the expressiveness of many graph database query languages. Bridging this gap is an important avenue for future work.

Extensions The first extension we introduce is the **union**.

Definition 2.11 (Union of Sets of Mappings) *Given two sets of mappings (like μ from Definition 2.8), we can create a new set of mappings by taking their union. This union operator is commutative and asso-*

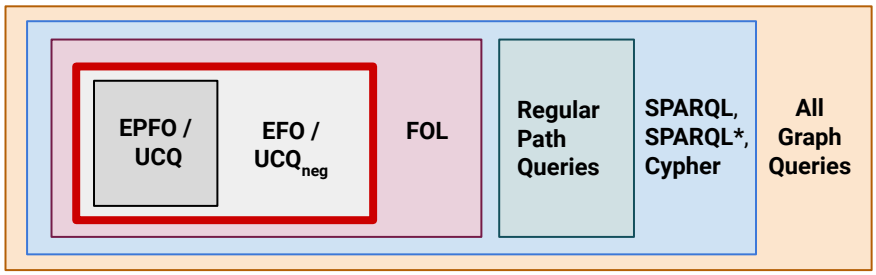

Figure 5: Current query answering models cover Existential Positive First Order (EPFO) and Existential First Order (EFO) logic fragments (marked in a red rectangle). EPFO and EFO are equivalent to unions of conjunctions (UCQ), and those with atomic negation (UCQ_{neg}), respectively. These, in turn, are a subset of first order logic (FOL). FOL queries, in turn, are only a subset of queries answerable by graph database languages. For example regular path queries cannot be expressed in FOL. Languages like SPARQL, SPARQL*, or Cypher, encompass all the query types and more.

ciative, we can hence also talk about the union of three or more mappings. It is permissible that the domains of the mappings in the input sets are not the same.

We can define a new type of query by applying the union operation on the outcomes of two or more underlying queries. If these underlying queries are basic graph queries, we will call this new type of queries **Unions of Conjunctive Queries with negation (UCQ_{neg})** and if the basic queries did not include negation, as **Union of Conjunctive Queries (UCQ)**. These classes are also familiar from FOL, and indeed correspond to a disjunction of conjunctions.

As an example, the following query is in UCQ_{neg} because it is a disjunction of conjunctive terms which consist of atoms a_i that are relation projections or their negations:

$$q = v_?.\exists v_1, \ldots, v_n : \Big(\underbrace{r_1(c, v_1)}_{a_1} \wedge \underbrace{r_2(v_1, v_2)}_{a_2}\Big) \vee \Big(\underbrace{\neg r_3(v_2, v_3)}_{a_3}\Big) \vee \cdots \vee \underbrace{r_k(v_n, v_?)}_{a_m}$$

Moreover, there are FOL fragments that are equivalent to these fragments. Specifically, all queries in **EPFO**, which are Existential Positive First Order sentences, have an equivalent query in UCQ, and all queries in **EFO**, Existential First Order sentences, have an equivalent query in UCQ_{neg}. The reason is that EPFO and EFO sentences can be written in the Disjunctive Normal Form (DNF) as a union of conjunctive terms.

Some query languages, like SPARQL, allow an optional part to a query. In our formalism, we can define the **optional** part using the union operator. Assuming there are n optional parts in the query, create 2^n different queries, in which other combinations of optional parts are removed. The answer to the query is then the union of the answers to all those queries. If the query language already allowed unions, then optionals do not make it more expressive.

Beyond these extensions, one could extend further to **all queries one could express with FOL**, which requires either universal quantification, or negation of conjunctions. These are, however, still not all possible graph queries. An example of interesting queries, which are not in FOL, are **conjunctive regular path queries**. These are akin to the path queries we discussed above, but without a specified length.

Definition 2.12 (Regular Path Query) *A regular path query is a 3-tuple (s, R, t), where $s \in$ `Term` the starting term of the path, $R \in$ `Term` the relation term of the path, and $t \in$ `Term` the ending term of the path. The query represents a path starting in s, traversing an arbitrary number of edges of type R ending in t.*

Because this kind of query is a conjunction of an arbitrary length, it cannot be represented in FOL. If one wants to express paths with a fixed length, this would be a multi-hop path like the one described above. If one wants to express a maximum length, then this could be done using a union of all allowed lengths. For the two latter cases, the query can still be expressed in $EPFO$.

Definition 2.13 (Regular Path Query Answering) *Given a knowledge graph $\mathcal{G}$ and a regular path query $\mathcal{Q} = (s, R, t)$. An answer to the query is any mapping $\mu : \texttt{Var}_Q \rightarrow \texttt{Con}$, such that if we replace all variables v in $\mathcal{Q}$ with $\mu(v)$, obtaining $(\hat{s}, \hat{R}, \hat{t})$, there exists a path in the graph that starts at node $\hat{s}$, then traverses one or more edges of type $\hat{R}$, and ends in $\hat{t}$. The set of all answers to the query is the set of all such mappings.*

There exist several variations on regular path queries and they can also be combined with the above query types to form new ones. In Fig. 5 we illustrate how the fragments relate to other query classes. Most methods fall strictly within the EFO fragment, *i.e.*, they only support a subset with restrictions as we discussed above. We will discuss further limitations of these methods in Section 6.

Two final aspects we want to highlight here are *projections* and *joins*.

Definition 2.14 (Projection of a Query Answer) *Given a query answer μ, and a set of variables $\mathcal{V} \in \texttt{Var}$. The projection of the query answer on the variables $\mathcal{V}$ is $\{(var, val | (var, val) \in \mu, var \in \mathcal{V}\}$.*

In other words, it is the answer but restricted to a specific set of variables. The query forms introduced above can all be augmented with a projection to only obtain values for specific variables. This corresponds to the `SELECT ?var` clause in SPARQL. Alternatively, it is possible to project all variables in $\mathcal{V}$ which is equivalent to the `SELECT *` clause. A query without any projected variable is a Boolean subgraph matching problem equivalent to the `ASK` query in SPARQL.

A join is used to combine the results of two separate queries into one. Specifically,

Definition 2.15 (Join of Query Answers) *Given two query answers μ_A and μ_B, and $\texttt{Var}_A$, and $\texttt{Var}_B$ the variables occurring in μ_A and μ_B, respectively. The join of these two answers only exists in case they have the same value for all variables they have in common, i.e., $\forall var \in \texttt{Var}_A \cap \texttt{Var}_B : \mu_A(var) = \mu_b(var)$. In that case, $\text{join}(\mu_A, \mu_B) = \mu_A \cup \mu_B$.*

Given two sets of answers, their join is defined as follows.

Definition 2.16 (Join of Query Answer Sets) *Given two sets of query answers A and B, the join of these two is a new set $\text{join}(A, B) = \{\text{join}(a, b) | a \in A, b \in B, \text{and } \text{join}(a, b) \text{ exists}\}$.*

This operation enables us to combine multiple underlying queries, potentially of multiple types into a single one. For example, given the set of answers from our example basic graph query above:

$$A = \{\{(?person, \texttt{Hinton}), (?uni, \texttt{UofT})\}, \{(?person, \texttt{Bengio}), (?uni, \texttt{UdeM})\}, \{(?person, \texttt{LeCun}), (?uni, \texttt{NYU})\}\}$$

and another set of answers

$$B = \{\{(?person, \texttt{Hinton}), (?born, 1947)\}, \{(?person, \texttt{Bengio}), (?born, 1964)\}, \{(?person, \texttt{Welling}), (?born, 1968)\}\}$$

The join of these becomes:

$$\text{join}(A, B) = \{\{(?person, \texttt{Hinton}), (?uni, \texttt{UofT}), (?born, 1947)\}, \{(?person, \texttt{Bengio}), (?uni, \texttt{UdeM}), (?born, 1964)\}\}$$

We will discuss joins further in Section 6.1, where we will use these basic building blocks to define a broader set of query operators, aiming to cover all operations that exist in SPARQL. This includes Kleene plus/star (+/*) for building property paths, FILTER, OPTIONAL, and different aggregation functions.

2.5 Approximate Graph Query Answering

Now that we have defined what Basic Approximate Graph Query Answering is, we can define what we mean by the more general Approximate Graph Query Answering. The definition of Approximate Graph Query Answering is the same as the basic case, but rather than only providing answers to basic queries, it is about answering a broader set of query types.

Definition 2.17 (Approximate Graph Query Answering) *Given a knowledge graph $\mathcal{G}$, subgraph of a complete, but not observable knowledge graph $\hat{\mathcal{G}}$, a query formalism,* ***any*** *graph query $\mathcal{Q}$ according to that formalism, and the scoring domain $\mathbb{R}$.*

An approximate graph query answer to the query $\mathcal{Q}$ is a function f which maps every possible mapping ($\mu : \texttt{Var}_Q \rightarrow \texttt{Con}$) to $\mathbb{R}$.

Note there that the variables are not always mapped to nodes which occur in the graph. It is well possible that the query contains an aggregation function which results in a literal value.

In the literature, we can find the concepts of *easy* and *hard* answers. This refers to whether the answers can be found by only having access to $\mathcal{G}$ or not.

Definition 2.18 (Easy and Hard answers) *Given a knowledge graph $\mathcal{G}$, subgraph of a larger unobservable graph $\hat{\mathcal{G}}$, and a query $\mathcal{Q}$. Easy and hard answers are defined in terms of exact query answering (Definition 2.10). The set of* ***easy*** *answers is the intersection of the answers obtained from $\mathcal{G}$, and those from $\hat{\mathcal{G}}$. The set of* ***hard*** *answers is the set difference between the answers from $\hat{\mathcal{G}}$ and those from $\mathcal{G}$.*

Note the asymmetry in the definitions. Easy answers are those that can be found in both $\mathcal{G}$ and $\hat{\mathcal{G}}$. Hard answers are those that can be found only in $\hat{\mathcal{G}}$ but not in $\mathcal{G}$. For Basic Graph Queries (Definition 2.7), all easy answers can also be found from $\hat{\mathcal{G}}$. However, for some more complex query types (*e.g.*, these which allow negation) there could also be answers which in $\mathcal{G}$ which are not in $\hat{G}$. We call these answers **false positives** in the context of answering over $\hat{\mathcal{G}}$.

2.6 Triangular Norms and Conorms

Answering logical queries implies execution of logical operators. Approximate query answering, in turn, implies continuous vector inputs and output truth values that are not necessarily binary. Besides, the methods often require that the logical operators are smooth and differentiable. Triangular norms (T-norms) and triangular conorms (T-conorms) define functions that generalize logical conjunction and disjunction, respectively, to the continuous space of truth values and implement fuzzy logical operations.

T-norm defines a continuous function $\top : [0,1] \times [0,1] \rightarrow [0,1]$ with the following properties $\top(x,y) = \top(y,x)$ (commutativity), $\top(x, \top(y,z)) = \top(\top(x,y),z)$ (associativity), and $y \leq z \rightarrow \top(x,y) \leq \top(x,z)$ (monotonicity). Also, we have 1 to be the identity element for $\top$, *i.e.*, $\top(x,1) = 1$. The goal of t-norms is to generalize logical conjunction with a continuous function. T-conorm can be seen as a duality of a t-norm that similarly defines a function $\perp$ with the same domain and range $\perp : [0,1] \times [0,1] \rightarrow [0,1]$. T-conorms use the continuous function $\perp$ to generalize disjunction to fuzzy logic. The function $\perp$ satisfies the same commutativity, associativity, and monotonicity properties as $\top$ with the only difference being 0 is the identity element, *i.e.*, $\perp(x,0) = x$.

There exist many triangular norms, conorms, and fuzzy negations (Klement et al., 2013; van Krieken et al., 2022) that stem from the corresponding logical formalisms, *e.g.*, (1) *Gödel logic* defines t-norm: $\top_{\min}(x,y) = \min(x,y)$, t-conorm: $\perp_{\max}(x,y) = \max(x,y)$; (2) *Product logic* with t-norm: $\top_{\text{prod}}(x,y) = x \cdot y$, t-conorm: $\perp_{\text{prod}}(x,y) = x + y - x \cdot y$; (3) in the *Łukasiewicz logic* t-norm: $\top_{\text{Łuk}}(x,y) = \max(x+y-1,0)$, t-conorm: $\perp_{\text{Łuk}}(x,y) = \min(x+y,1)$. Using fuzzy negation $N(x) = 1 - x$, it is easy to verify that $\perp(x,y) = N(\top(N(x),N(y)))$ (De Morgan's laws) naturally obtaining a pair of $(\top, \perp)$.

2.7 Graph Representation Learning

Graph Representation Learning (GRL) is a subfield of machine learning aiming at learning low-dimensional vector representations of graphs or their elements such as single nodes (Hamilton, 2020). For example, $\mathbf{h}_v \in \mathbb{R}^d$ denotes a d-dimensional vector associated with a node v. Conceptually, we want nodes that share certain structural and semantic features in the graph to have similar vector representations (where similarity is often measured by a distance function or its modifications).

Shallow Embeddings The first GRL approaches focused on learning shallow node embeddings, that is, learning a unique vector per node directly used in the optimization task. For homogeneous (single-relation) graphs, DeepWalk (Perozzi et al., 2014) and node2vec (Grover & Leskovec, 2016) trained node embeddings on the task of predicting walks in the graph whereas in multi-relational graphs TransE (Bordes et al., 2013) trained node and edge type embeddings in the autoencoder fashion by reconstructing the adjacency matrix.

Graph Neural Networks The idea of *graph neural networks* (GNNs) (Scarselli et al., 2008) implies learning an additional neural network encoder with shared parameters on top of given (or learnable) node features by performing neighborhood aggregation. This framework can be generalized to *message passing* (Gilmer et al., 2017) where at each layer t a node v receives messages from its neighbors (possibly adding edge and graph features), aggregates the messages in the permutation-invariant way, and updates the representation:

$$\mathbf{h}_v^{(t)} = \textsc{Update}\Big(\mathbf{h}_v^{(t-1)}, \textsc{Aggregate}_{u\in\mathcal{N}(v)}\big(\textsc{Message}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}, \mathbf{e}_{uv})\big)\Big)$$

Here, $\mathbf{h}_u$ is a feature of the neighboring node u, $\mathbf{e}_{uv}$ is the edge feature, Message function builds a message from node u to node v and can be parameterized with a neural network. As the set of neighbors $\mathcal{N}(v)$ is unordered, Aggregate is often a permutation-invariant function like *sum* or *mean*. The Update function takes the previous node state and aggregated messages of the neighbors to produce the final state of the node v at layer t and can be parameterized with a neural network as well.

Classical GNN architectures like GCN (Kipf & Welling, 2017), GAT (Velickovic et al., 2018), and GIN (Xu et al., 2019) were designed to work with homogeneous, single-relation graphs. Later, several works have developed GNN architectures that work on heterogeneous graphs with multiple relations (Schlichtkrull et al., 2018; Vashishth et al., 2020; Zhu et al., 2021). GNNs and message passing paved the way for *Geometric Deep Learning* (Bronstein et al., 2021) that leverages symmetries and invariances in the input data as inductive bias for building deep learning models.

3 Neural Graph Databases

Traditional databases (including graph databases) are designed around two crucial modules: the storage layer for data and the query engine to process queries over the stored data. From that perspective, neural database *per se* is not a novel term and many machine learning systems already operate in this paradigm when data is encoded into model parameters and querying is equivalent to a forward pass that can output a new representation or prediction for a downstream task. One of the first examples of neural databases is *vector databases*. In vector databases, the storage module consists of domain-agnostic vector representations of the data, which can be muliti-modal *e.g.*, text paragraphs or images. Vector databases belong to the family of storage-oriented systems commonly built around approximate nearest neighbor libraries (ANN) like Faiss (Johnson et al., 2019) or ScaNN (Guo et al., 2020) to answer distance-based queries (like maximum inner product search, MIPS). Being encoder-independent (that is, any encoder yielding vector representations can be a source), vector databases lack graph reasoning and complex query answering capabilities. Still, ANN systems are a convenient choice for implementing certain layers of Neural Graph Databases as we describe below.

With the recent rise of large-scale pretrained models (*i.e.*, *foundation models* (Bommasani et al., 2021)), we have witnessed their huge success in natural language processing and computer vision tasks. We argue that such foundation models are also a prominent example of neural databases. In foundation models, the "storage module" might be presented directly with model parameters or outsourced to an external index often used in retrieval-augmented models (Lewis et al., 2020; Guu et al., 2020; Alon et al., 2022) since encoding all world knowledge even into billions of model parameters is hard. The "query module" performs in-context learning either via filling in the blanks in encoder models (BERT or T5 style) or via prompts in decoder-only models (GPT-style) that can span multiple modalities, *e.g.*, learnable tokens for vision applications (Kolesnikov et al., 2022) or even calling external tools (Mialon et al., 2023). Applied to the text modality, Thorne et al. (2021a;b) devise *Natural Language Databases (NLDB)* where atomic elements are textual facts encoded to a

vector via a pre-trained language model (LM). Queries to NLDB are sent as natural language utterances that get encoded to vectors and query processing employs the *retriever-reader* approach. First, a dense neural *retriever* returns a support set of candidate facts, and a fine-grained neural *reader* performs a *join* operation over the candidates (as a sequence-to-sequence task).

The amount of graph-structured data is huge and spans numerous domains like general knowledge with Freebase (Bollacker et al., 2008), DBpedia (Lehmann et al., 2015), Wikidata (Vrandecic & Krötzsch, 2014), YAGO (Pellissier Tanon et al., 2020), commonsense knowledge with ConceptNet (Speer et al., 2017), ATOMIC (Hwang et al., 2021), and biomedical knowledge such as Bio2RDF (Callahan et al., 2013) and PrimeKG (Chandak et al., 2023). With the growing sizes, the incompleteness of those graphs grows simultaneously. At this scale, symbolic methods struggle to provide a meaningful approach to deal with incompleteness. Therefore, we argue that neural reasoning and graph representation learning methods are capable of addressing incompleteness at scale while maintaining high expressiveness and supporting complex queries beyond simple link prediction. We propose to study those methods under the framework of *Neural Graph Databases (NGDB)*. The concept of NGDB extends the ideas of neural databases to the graph domain. NGDB combines the advantages of traditional graph databases (graphs as a first-class citizen, efficient storage, and uniform querying interface) with modern graph machine learning (geometric and physics-inspired vector representations, ability to work with incomplete and noisy inputs by default, large-scale pre-training and fine-tuning on downstream applications). In contrast to the work of Besta et al. (2022) that proposed LPG2Vec, a featurization strategy for labeled property graphs to be then used in standard graph database pipelines, we design the NGDB concept to be *neural-first*.

Using the definition of Approximate Graph Query Answering (Definition 2.17), we define Neural Graph Databases as follows.

Definition 3.1 (Neural Graph Database, NGDB) *A Neural Graph Database (see Fig. 6) is a tuple (S, E, f_θ). S is a Neural Graph Storage (see Section 3.1), E is a Neural Query Engine (see Section 3.2), and f_θ is a parameterized Approximate Graph Query Answering function, where θ represents a set of parameters.*

In particular, our NGDB design agenda includes:

- The **data incompleteness assumption**, *i.e.*, the underlying data might have missing information on node-, link-, and graph-levels which we would like to infer and leverage in query answering;
- **Inductiveness and updatability**, *i.e.*, similar to traditional databases that allow updates and instant querying, representation learning algorithms for building graph latents have to be inductive and generalize to unseen data (new entities and relation at inference time) in the zero-shot (or few-shot) manner to prevent costly re-training (for instance, of shallow node embeddings);
- **Expressiveness**, *i.e.*, the ability of latent representations to encode logical and semantic relations in the data akin to FOL (or its fragments) and leverage them in query answering. Practically, the set of supported logical operators for neural reasoning should be close to or equivalent to standard graph database languages like SPARQL or Cypher;
- **Multimodality** beyond KGs, *i.e.*, any graph-structured data that can be stored as a node or record in classical databases (consisting, for example, of images, texts, molecular graphs, or timestamped sequences) and can be imbued with a vector representation is a valid source for the Neural Graph Storage and Neural Query Engine.

The key methods to address the NGDB design agenda are:

- **Vector representation as the atomic element**, *i.e.*, while traditional graph DBs hash the adjacency matrix (or edge list) in many indexes, the incompleteness assumption implies that both given edges **and** graph latents (vector representations) become the *sources of truth* in the *Neural Graph Storage*;

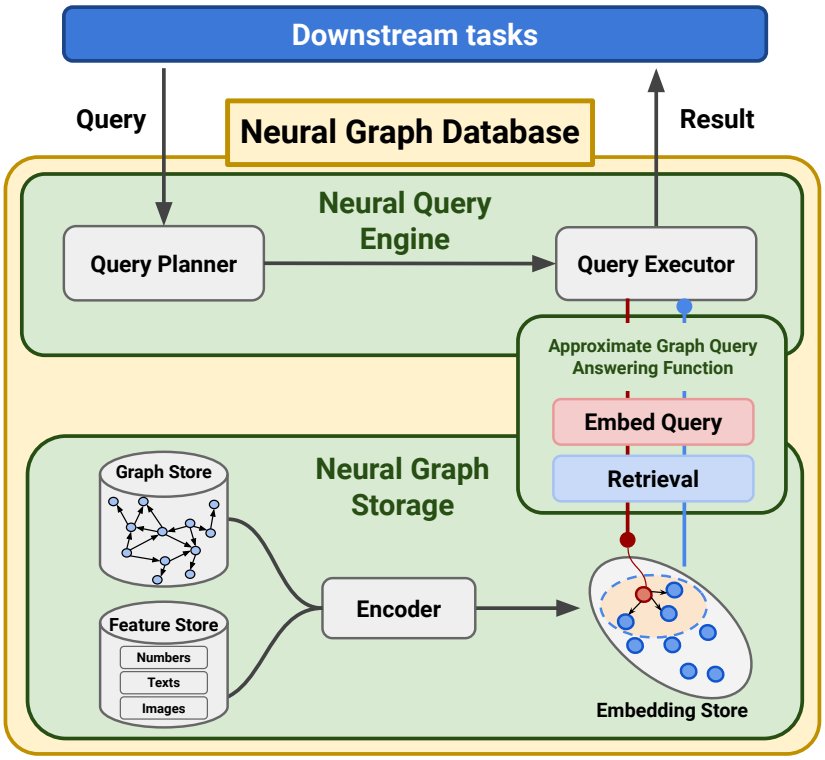

Figure 6: A conceptual scheme of Neural Graph Databases. An input query is processed by the *Neural Query Engine* where the *Planner* derives a computation graph of the query and the *Executor* executes the query in the latent space. *Neural Graph Storage* employs *Graph Store* and *Feature Store* to obtain latent representations in the *Embedding Store*. The *Executor* communicates with the embedding store via the *Approximate Graph Query Answering Function* f_θ *(Definition 3.1)* to retrieve and return results.

- **Neural query execution in the latent space**, *i.e.*, basic operations such as edge traversal cannot be performed solely symbolically due to the incompleteness assumption. Instead, the *Neural Query Engine* operates on both adjacency and graph latents to incorporate possibly missing data into query answering;

A conceptual scheme of NGDB is presented in Fig. 6. On a higher level, NGDB contains two main components, *Neural Graph Storage* and *Neural Query Engine* that we describe in the following Section 3.1 and Section 3.2, respectively. The processing pipeline starts with the query sent by some application or downstream task already in a structured format (obtained, for example, via semantic parsing (Drozdov et al., 2022) if an initial query is in natural language). The query first arrives to the *Neural Query Engine*, and, in particular, to the *Query Planner* module. The task of the Query Planner is to derive an efficient computation graph of atomic operations (*e.g.*, projections and logical operations) with respect to the query complexity, prediction tasks, and underlying data storage such as possible graph partitioning. We elaborate on similarities and differences of the planning mechanism to standard query planners in classic databases in Section 3.2. The derived plan is then sent to the *Query Executor* that encodes the query in a latent space, executes the atomic operations over the underlying graph and its latent representations, and aggregates the results of atomic operations into a final answer set. The execution is done via the *Retrieval* module that communicates with the *Neural Graph Storage*. The storage layer consists of (1) *Graph Store* for keeping the multi-relational adjacency matrix in space- and time-efficient manner (*e.g.*, in various sparse formats like COO and CSR; (2) *Feature Store* for keeping node- and edge-level multimodal features associated with the underlying graph; (3) *Embedding Store* that leverages an *Encoder* module to produce graph representations in a latent space based on the underlying adjacency and associated features. The Retrieval module queries

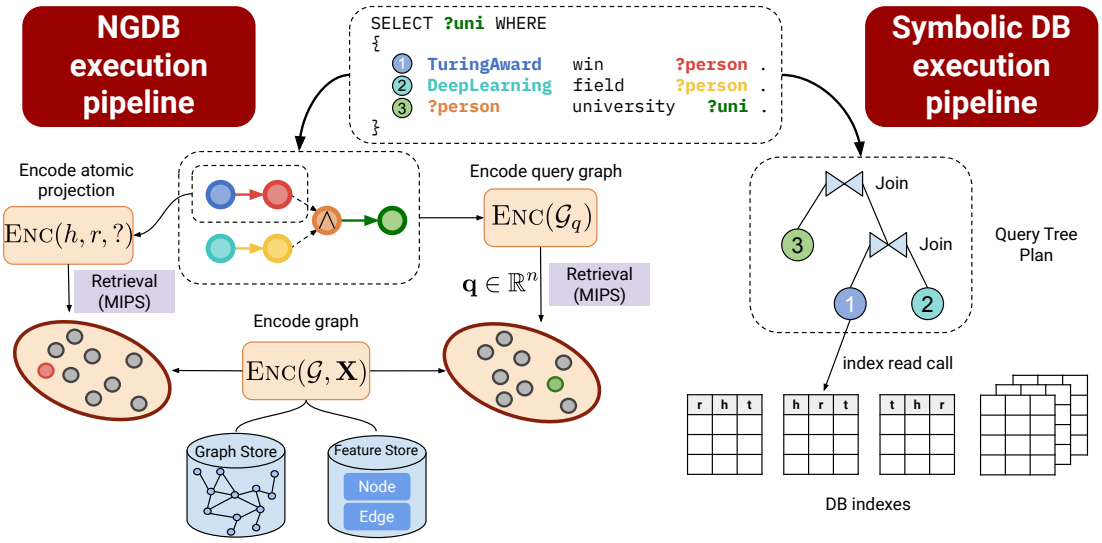

Figure 7: The storage and execution pipeline of NGDB (left) and traditional databases (right). Traditional DBs store the graph in a collection of lookup indexes and each query pattern from the query tree plan is answered by some of those indexes. In NGDBs, due to graph incompleteness, the graph and its features are first encoded in a latent space. Queries (or their atomic patterns) are encoded in a latent space either and probed using the Retrieval module (*e.g.*, can be implemented with MIPS).

the encoded graph representations to build a distribution of potential answers to atomic operations. In the following subsections, we describe the Neural Graph Storage and Neural Query Engine in more detail.

3.1 Neural Graph Storage

In traditional graph databases, storage design often depends on the graph modeling paradigm. The two most popular paradigms are Resource Description Framework (RDF) graphs (Brickley et al., 2014) and Labeled Property Graphs (LPG) (Besta et al., 2022). While the detailed comparison between those frameworks is out of scope of this work, the principal difference consists in that RDF is a triple-based model allowing for some formal *semantics* suitable for symbolic reasoning whereas LPG allows literal attributes over edges but has no formal semantics. We posit that the new RDF-star paradigm (Hartig et al., 2022) would be a convergence point of graph modeling combining the best of both worlds, *i.e.*, attributes over edges (enabling other nodes and relations to be in key-value attributes) together with expressive formal semantics. We note that hyper-relational KGs (Definition 2.3) and Wikidata Statement Model (Vrandecic & Krötzsch, 2014) are conceptually close to the RDF-star paradigm. On a physical level, graph databases store edges employing various indexes and data structures optimized for read (*e.g.*, B+ trees (Neumann & Weikum, 2008) or HDT (Fernández et al., 2013)) or write applications (such as LSM trees (Sagi et al., 2022)).

However, unlike the above methods, we propose the concept of a *neural* graph storage (Fig. 7) where both the input graph and its vector representations are sources of truth. Physically, it consists of the (1) *Graph Store* that stores a (multi-relational) adjacency matrix that in the basic form can be implemented with sparse COO, CSR, or more efficient compressed formats, (2) *Feature Store* that stores node- and edge-level feature vectors of various formats and modalities, *e.g.*, numerical, categorical, text data, or already given vectors. (3) *Embedding Store* that leverages an *Encoder* to produce graph representations in a latent space based on the underlying adjacency and associated features. The embedding store is one of the biggest differences between neural graph storage and its counterpart in traditional graph DBs. The embedding process can be viewed as a compression step but the semantic and structure similarity of entities/relations is kept. The distance between entities/relations in the embedding space should be positively correlated with the semantic/structure similarity. There are several options for the architecture of the encoder.

First, we may implement the encoder as a matrix lookup, of which each row corresponds to the embedding of an entity/relation. The benefit of such modeling is that it provides the most flexibility and the most number of free parameters – any entry of the embedding matrix is trainable. However, it comes at the cost that it is challenging to edit the storage. We cannot easily add new entities/relations to the storage because any novel entity/relation may need a new embedding that requires learning from scratch. If we remove an entity and later add it back, such embedding matrix modeling also cannot recover the original learned embedding.

Another option for the encoder is a graph neural network (GNN) that can be analyzed through the lens of message passing and neighborhood aggregation (Section 2.7). The idea is that we may learn the entity/relation embeddings by aggregating its neighbor information. The GNN parameters are shared for all the entities and relations on the graph. Such modeling provides clear benefit with much better generalization capability then shallow embedding matrices.

One important process in the neural graph storage is the **retrieval** step. Traditional graph databases directly perform identity-based exact match retrieval from the indexes. In contrast, in a neural graph storage, since we store each entity and relation in the latent space, besides performing retrieval by the entity/relation id, users can also input an vector, and the retrieval process may return "relevant" entities/relations by measuring the distance in the embedding space. The retrieval process can be seen as a nearest neighbor search of the input vector in the embedding space. There are three direct benefits of a neural graph storage compared with traditional storages. The first advantage is that since the retrieval is operated in the embedding space with a predefined distance function, each retrieved item naturally comes with a score which may represent the confidence/relevance of the input vector in a retrieval step. Besides, NGDB allows for different definitions of the latent space and the distance function (which will be detailed in Section 5.2), such that NGDB is flexible and users may customize the latent space and distance function based on different desired properties and user need. Lastly, with the whole literature on efficient nearest neighbor search, we have the opportunity to implement the retrieval step on extremely large graphs with billions of nodes and edges with high efficiency and scalability. Existing frameworks including Faiss (Johnson et al., 2019), ScaNN (Guo et al., 2020) provide scalable implementation of nearest neighbor search. A major limitation, however, is that existing frameworks are only applicable to L1, L2 and cosine distance functions. It is still an open research problem how to design efficient scalable nearest neighbor search algorithms for more complex distance functions such as KL divergence so that we can retrieve with much better efficiency for different CLQA methods.

3.2 Neural Query Engine

In traditional databases, a typical query engine (Neumann & Weikum, 2008; Endris et al., 2018) performs three major operations. (1) Query parsing to verify syntax correctness (often enriched with a deeper semantic analysis of query terms); (2) Query planning and optimization to derive an efficient query plan (usually, a tree of relational operators) that minimizes computational costs; (3) Query execution that scans the storage and processes intermediate results according to the query plan.

In NGDBs, the sources of truth are both graph adjacency (possibly incomplete) and latent graph representations (*e.g.*, vector representations of entities and relations). Similarly, queries (or their atomic operations) are encoded into a latent space. To answer encoded queries over latent spaces, we devise *Neural Query Engines* that include two modules: (1) *Query Planner* to derive an efficient query plan of atomic operations (*e.g.*, projections and logical operators) maximizing completeness (all answers over existing edges must be returned) and inference (of missing edges predicted on the fly) taking into account query complexity, prediction tasks, and partitioning of sources; (2) *Query Executor* that encodes the derived plan in a latent space, executes the atomic operations (or the whole query) over the graph and its latent representations, and aggregates intermediate results into a final answer set with possible postprocessing.

Broadly in Deep Learning, prompting large language models (LLMs) can be seen as a form of a neural query engine where queries are unstructured texts. Recent prompting techniques like Chain of Thought (Wei et al., 2022) or Program of Thought (Chen et al., 2022a) achieved remarkable progress in natural language reasoning (Qiao et al., 2022) by providing few prompts with "common sense" examples of solving a given problem step-by-step. Such step-by-step instructions resemble a *query plan* designed manually by a prompt engineer and optimized for *query execution* against a particular LLM where query execution is framed as the

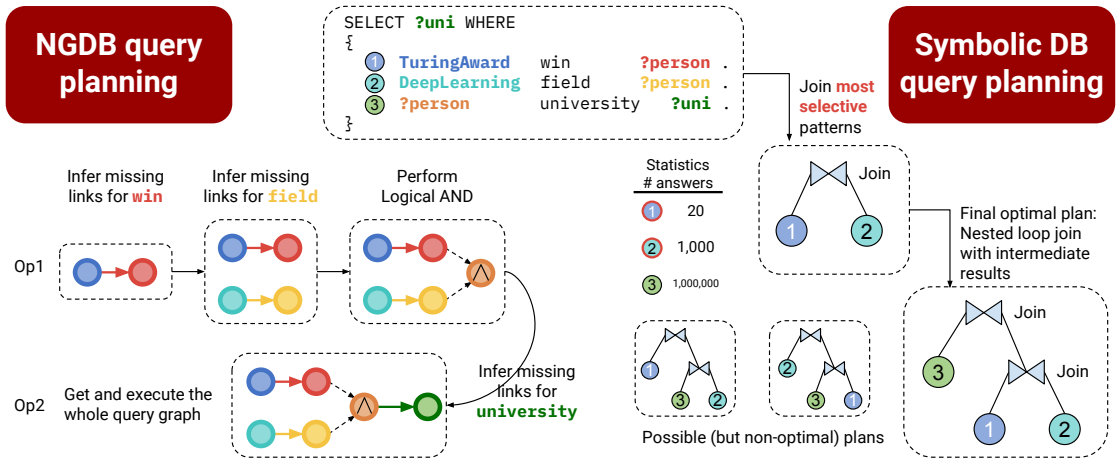

Figure 8: Query planning in NGDBs (left) and traditional graph DBs (right). The NGDB planning (assuming incomplete graphs) can be performed autoregressively step-by-step (1) or generated entirely in one step (2). The traditional DB planning is cost-based and resorts to metadata (assuming complete graphs and extracted from them) such as the number of intermediate answers to build a tree of join operators.

language model objective (predicting next few tokens). Prompting often does not require any additional LLM fine-tuning or training and works in the inference regime (often called *in-context learning*). Pre-trained LLMs can thus be seen as *neural databases* that can be queried with a sequence of prompts organized in a domain-specific query language (Beurer-Kellner et al., 2022). Recently, a similar prompting technique (Drozdov et al., 2022) demonstrated strong systematic generalization skills by solving a long-standing semantic parsing task of converting natural language questions to SPARQL queries. Several major drawbacks of such querying techniques are: (1) a limited context window (usually, less than 8192 tokens) that does not allow prompting the whole database content; (2) issues with factual correctness and hallucinating of the generated reponses.

Query Planner In NGDBs, an input query is expected to arrive in the structured form, *e.g.*, as generic as FOL – we leave the discussion on possible query languages out of scope of this work emphasizing the breadth of general and domain-specific languages. For illustrating example queries, however, we use the SPARQL notation as one of the standard query languages for graph databases. Broadly, principal differences between symbolic and neural query planning are illustrated in Fig. 8. While traditional graph DBs are symbolic and deductive, NGDBs are neural and inductive (*i.e.*, can generalize to unseen data). As we assume the graph data is incomplete, neural reasoning is expected to process more intermediate results that makes query planning even more important in deriving the most efficient plan.

The task of the Query Planner is to optimize the execution process by deriving the most efficient query plan, *i.e.*, the execution order of atomic and logical operations, given the query complexity, expected prediction task, and configuration of the storage. We envision the storage configuration to play an important role in very large graphs that cannot be stored entirely in the main memory. A typical configuration would resort to partitioning of graphs and latent representations across many devices in a distributed fashion. Therefore, the planner has to identify which parts of the input query to send to a relevant partition. Training and inference of ML models in distributed environments (with possible privacy restrictions) is at the core of *federated learning* (Li et al., 2020) and *differential privacy* (Dwork et al., 2006) and we posit they would be important components of very large NGDBs.

Common metrics of evaluating the quality of plans are *execution time* (in units of time) and *throughput* (number of queries per unit of time). In traditional graph DBs, different query plans return the same answers but require different execution time (Endris et al., 2018) due to the order of join operators (*e.g.*, nested loop joins or hash joins), the number of calls to indexes, and intermediate results to process. In contrast, NGDBs execute queries in the latent space together with link prediction and, depending on the

approach, time complexity of atomic operations (relation projection and logical operators) might depend on the hidden dimension $\mathcal{O}(d)$ or the number of entities $\mathcal{O}(\mathcal{E})$ (we elaborate on complexity in Section 5.4).

To date, query planning is still a challenge for neural query answering methods as all existing approaches (Section 5) execute an already given sequence of operators and all existing datasets (Section 7) provide such a hardcoded sequence without optimizations. Furthermore, some existing approaches, *e.g.*, CQD (Arakelyan et al., 2021), are susceptible to the issue when changing a query plan might result in a different answer set.

We hypothesize that principled approaches for deriving efficient query plans taking into account missing edges and incomplete graphs might be framed as *Neural Program Synthesis* (Nye et al., 2020) often used to derive query plans for LLMs to solve complex numerical reasoning tasks (Gupta et al., 2020; Chen et al., 2020). In the complex query answering domain, the first attempt to apply neural program synthesis for generating query plans was made by the Latent Execution-Guided Reasoning (LEGO) framework (Ren et al., 2021). LEGO iteratively expands the initial query tree root by sampling from the space of relation projection and logical operators. The sampling is based on learnable heuristics and pruning mechanisms.

Query Executor Once the query plan is finalized, the *Query Executor* module encodes the query (or its parts) into a latent space, communicates with the Graph Storage and its Retrieval module, and aggregates intermediate results into the final answer set. Following the Definition 2.18, there exists easy answers and hard answers. Query Executors are expected to return the compelte set of easy answers and recover missing edges to return the missing hard answers.

There exist two common mechanisms for neural query execution described in Section 5: (1) *atomic*, resembling traditional DBs, when a query plan is executed sequentially by encoding atomic patterns (such as relation projections), retrieving their answers, and executing logical operators as intermediate steps; (2) *global*, when the entire query graph is encoded and executed in a latent space in one step. For example (Fig. 7), given a query plan $q = U_? . \exists V : \mathit{win}(\texttt{TuringAward}, V) \wedge \mathit{field}(\texttt{DeepLearning}, V) \wedge \mathit{university}(V, U_?)$, the atomic mechanism executes separate relation projections, *e.g.*, $\mathit{win}(\texttt{TuringAward}, V)$, sequentially while the global mechanism encodes the whole query graph and probes it against the latent space of the graph.

Direct implications of the chosen mechanism include computational complexity (Section 5.4) and supported logical operators (Section 6), *i.e.*, fully latent mechanisms are mostly limited to conjunctive and intersection queries and have worse generalization qualities. To date, most methods follow the atomic mechanism.

The main challenge for neural query execution is matching query expressiveness to that of symbolic languages like SPARQL or Cypher. The challenge includes three aspects: (1) handling more expressive query operators such as `FILTER` or aggregations like `COUNT` or `SUM`; (2) supporting query patterns more complex than trees; (3) supporting several projected variables and operations on them. We elaborate in Section 6.

A Taxonomy of Query Reasoning Methods In the following sections, we devise a taxonomy of query answering methods as a component of the *Neural Query Engine (NQE)*. We categorize existing and future approaches along three main directions: (i) **Graphs** – what is the underlying structure against which we answer queries; (ii) **Modeling** – how we answer queries and which inductive biases are employed; (iii) **Queries** – what we answer, what are the query structures and what are the expected answers. The taxonomy is presented in Fig. 9. In the following sections, we describe each direction in more detail and illustrate them with examples covering the whole existing literature on complex query answering (more than 40 papers).

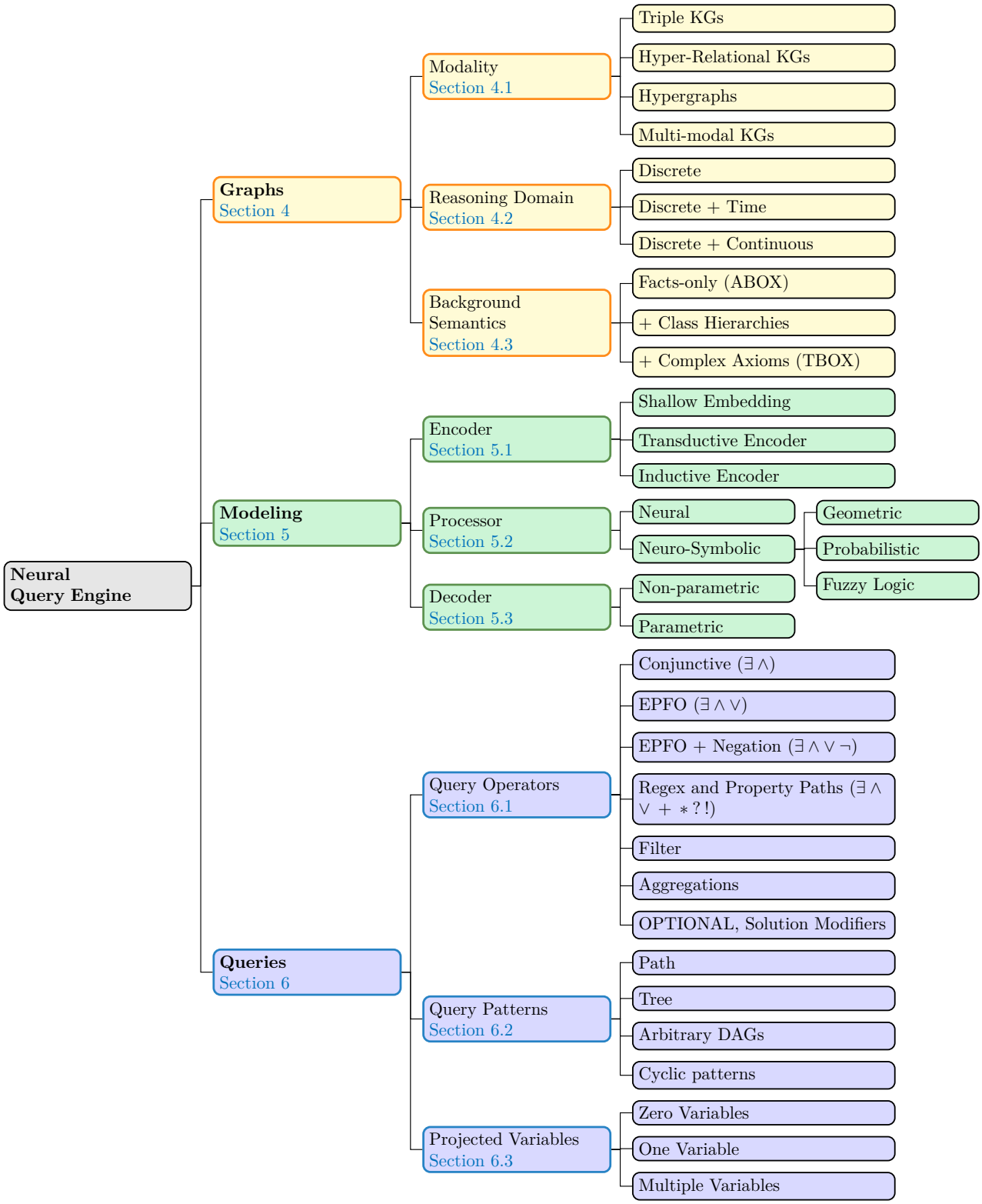

Figure 9: The Neural Query Engine Taxonomy consists of three main branches – Graphs, Modeling, and Queries. We describe each branch in more detail including prominent examples in the relevant sections.

4 Graphs

The *Graphs* category covers the underlying graph structure ($\mathcal{G}$ in Definition 2.9) against which complex queries are sent and the answers are produced. Understanding the graph, its contents and modeling paradigms is crucial for designing query answering models. To this end, we propose to analyze the underlying graph from three aspects: *Modality*, *Reasoning Domain*, and *Background Semantics*.

4.1 Modality

We highlight four modalities common for KGs and graph databases: standard **triple-based KGs** adhering to the RDF data model, **hyper-relational KGs** following the RDF-star or Labeled Property Graph (LPG) formats, and **hypergraph KGs** (we elaborate on choosing an appropriate modeling paradigm in Section 3.1). The difference among the three modalities is illustrated in Fig. 2. Additionally, we outline **multi-modal KGs** that contain not just a graph of nodes and edges, but also text, images, audio, video, and other data formats linked to the underlying graph explicitly or implicitly.

We categorize the literature along the Modality aspect in Table 2. To date, most query answering approaches operate solely on **triple-based** graphs. Among approaches supporting **hyper-relational** graphs, we are only aware of StarQE (Alivanistos et al., 2022) that incorporates entity-relation *qualifiers* over labeled edges and its extension NQE (Luo et al., 2023). We posit that the hyper-relational model might serve as a theoretical foundation of temporal query answering approaches since temporal attributes are in fact continuous key-value edge attributes. To date, we are not aware of complex query answering models supporting **hypergraphs** or **multi-modal** graphs. We foresee them as possible area of future research in the area.

Table 2: Complex Query Answering approaches categorized under *Modality*.

Triple-only	Hyper-Relational	Hypergraph	Multi-modal
GQE (Hamilton et al., 2018), GQE w hash (Wang et al., 2019), CGA (Mai et al., 2019), TractOR (Friedman & Broeck, 2020), Query2Box (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), EmQL (Sun et al., 2020), MPQE (Daza & Cochez, 2020), Shv (Gebhart et al., 2021), Q2B Onto (Andresel et al., 2021), RotatE-Box (Adlakha et al., 2021), BiQE (Kotnis et al., 2021), HyPE (Choudhary et al., 2021b), NewLook (Liu et al., 2021), CQD (Arakelyan et al., 2021), PERM (Choudhary et al., 2021a), ConE (Zhang et al., 2021b), LogicE (Luus et al., 2021), MLPMix (Amayuelas et al., 2022), FuzzQE (Chen et al., 2022b), GNN-QE (Zhu et al., 2022), GNNQ (Pflueger et al., 2022), SMORE (Ren et al., 2022), KGTrans (Liu et al., 2022), LinE (Huang et al., 2022b), Query2Particles (Bai et al., 2022a), TAR (Tang et al., 2022), TeMP (Hu et al., 2022), FLEX (Lin et al., 2022b), TFLEX (Lin et al., 2022a), NodePiece-QE (Galkin et al., 2022b), ENeSy (Xu et al., 2022), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022), QTO (Bai et al., 2022b), SignalE (Wang et al., 2022a), LMPNN (Wang et al., 2023c), Var2Vec (Wang et al., 2023a), CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023), Query2Geom (Sardina et al., 2023), SQE (Bai et al., 2023)	StarQE (Alivanistos et al., 2022), NQE (Luo et al., 2023)	None	None

4.2 Reasoning Domain

Following Definition 2.8, a query $\mathcal{Q}$ includes constants $\texttt{Con}$, variables $\texttt{Var}$, and returns *answers* as mappings $\mu : \texttt{Var}_Q \rightarrow \texttt{Con}$. By *Reasoning Domain* we understand a space of possible constants, variables, and answers, that is, what query answering models can reason about. We highlight three common domains (*Discrete*, *Discrete + Time*, *Discrete + Continuous*), illustrate them in Fig. 10, and categorize existing works in Table 3. Each subsequent domain is a superset of the previous domains, *e.g.*, *Discrete + Continuous* includes the capabilities of *Discrete* and *Discrete + Time* and expands the space to continuous inputs and outputs.

In the *Discrete* domain, constants, variables, and answers can be entities $\mathcal{E}$ and (or) relation types $\mathcal{R}$ of the KG, $\texttt{Con} \subseteq \mathcal{E} \cup \mathcal{R}$, $\texttt{Var} \subseteq \mathcal{E} \cup \mathcal{R}$, $\mu \subseteq \mathcal{E} \cup \mathcal{R}$. That is, input queries may only contain entities (or relations) and their answers are only entities (or relations). For example, a query $?x : \texttt{education}(\texttt{Hinton}, x)$ in Fig. 10 can only return two entities $\{\texttt{Edinburgh}, \texttt{Cambridge}\}$ as answers. Conceptually, the framework allows relation types $r \in \mathcal{R}$ to be variables as well describing, for example, a SPARQL graph pattern

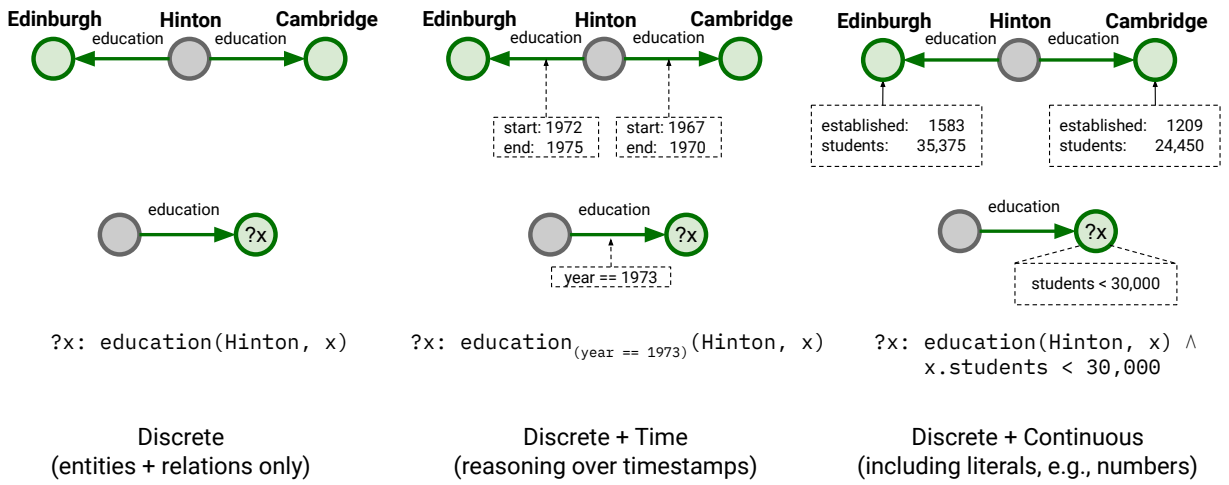

Figure 10: Reasoning Domains. The *Discrete* domain only allows entities and relations as constants, variables, and answers. The *Discrete + Time* domain extends the space to discrete timestamps and time-specific operators. The *Discrete + Continuous* domain allows continuous inputs (literals) and outputs.

$\{\texttt{Hinton ?r Cambridge}\}$ – or $?r : r(\texttt{Hinton}, \texttt{Cambridge})$ in the functional form – that returns all relation types between two nodes `Hinton` and `Cambridge`. However, to the best of our knowledge, the majority of existing query answering literature and datasets limit the space of constants, variables, and answers to entities-only, $\texttt{Con} \subseteq \mathcal{E}$, $\texttt{Var} \subseteq \mathcal{E}$, $\mu \subseteq \mathcal{E}$ and all relation types are given in advance (we discuss queries structure in more detail in Section 6). To date, most of the literature in the field belongs to the *Discrete* reasoning domain (Table 3).

Some nodes and edges might have *timestamps* from a set of discrete timestamps $t \in \mathcal{TS}$ indicating a validity period of a certain statement. In a more general case, certain subgraphs might be timestamped. We define the *Discrete + Time* domain when queries include temporal data. In this domain, the set of constants is extended with timestamps $\texttt{Con} \subseteq \mathcal{E} \cup \mathcal{R} \cup \mathcal{TS}$ and relation projections might be instantiated with a certain timestamp $R_t(a, b)$. For instance (Fig. 10), given a timestamped graph and a query $?x : \texttt{education}_{\texttt{year}==1973}(\texttt{Hinton}, x)$, the answer set includes only `Edinburgh` as the timestamp 1973 falls into the validity period of only one edge.

It is possible to extend the domain of variables and query answers with timestamps as well, $\texttt{Var}, \mu \subseteq \mathcal{E} \cup \mathcal{R} \cup \mathcal{TS}$, such that queries might employ timestamps as intermediate existentially quantified variables and possible answers can be entities or timestamps. This approach is followed by the Temporal Feature-Logic Embedding Framework (TFLEX) by Lin et al. (2022a) that defines additional operators **before**, **after**, **between** over edges with discrete timestamps.

Finally, the most expressive domain is *Discrete + Continuous* that enables reasoning over continuous inputs (such as numbers, texts, continuous timestamps) often available as node and edge attributes or *literals*. Formally, for numerical data, the space of constants, variables, and answers is extended with real numbers $\mathbb{R}$, *i.e.*, $\texttt{Con}, \texttt{Var}, \mu \subseteq \mathcal{E} \cup \mathcal{R} \cup \mathcal{TS} \cup \mathbb{R}$. An example query in Fig. 10 $?x : \texttt{education}(\texttt{Hinton}, x) \wedge x.\texttt{students} < 30000$ includes a conjunctive term $x.\texttt{students} < 30000$ that requires numerical reasoning over the `students` attribute of a variable x to produce the answer `Cambridge`. In a similar fashion, extending the answer set to continuous outputs can be framed as a regression task. To date, we are not aware of any complex query answering approaches capable of working in the continuous domain. Still, the ability to reason over continuous data is crucial for query answering given that most real-world KGs heavily rely on literals.

4.3 Background Semantics

Relational databases often contain a *schema*, that is, a specification of how tables and columns are organized that gives a high-level overview of the database content. In graphs databases, schemas exist as well and

Table 3: Complex Query Answering approaches categorized under *Reasoning Domain*.

Discrete	Discrete + Time	Discrete + Continuous
GQE (Hamilton et al., 2018), GQE w hash (Wang et al., 2019), CGA (Mai et al., 2019), TractOR (Friedman & Broeck, 2020), Query2Box (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), EmQL (Sun et al., 2020), MPQE (Daza & Cochez, 2020), Shv (Gebhart et al., 2021), Q2B Onto (Andresel et al., 2021), RotatE-Box (Adlakha et al., 2021), BiQE (Kotnis et al., 2021), HyPE (Choudhary et al., 2021b), NewLook (Liu et al., 2021), CQD (Arakelyan et al., 2021), PERM (Choudhary et al., 2021a), ConE (Zhang et al., 2021b), LogicE (Luus et al., 2021), MLPMix (Amayuelas et al., 2022), FuzzQE (Chen et al., 2022b), GNN-QE (Zhu et al., 2022), GNNQ (Pflueger et al., 2022), SMORE (Ren et al., 2022), KGTrans (Liu et al., 2022), LinE (Huang et al., 2022b), Query2Particles (Bai et al., 2022a), TAR (Tang et al., 2022), TeMP (Hu et al., 2022), FLEX (Lin et al., 2022b), NodePiece-QE (Galkin et al., 2022b), ENeSy (Xu et al., 2022), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022), StarQE (Alivanistos et al., 2022), QTO (Bai et al., 2022b), SignalE (Wang et al., 2022a), LMPNN (Wang et al., 2023c), NQE (Luo et al., 2023), Var2Vec (Wang et al., 2023a), CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023), Query2Geom (Sardina et al., 2023), SQE (Bai et al., 2023)	TFLEX (Lin et al., 2022a)	None

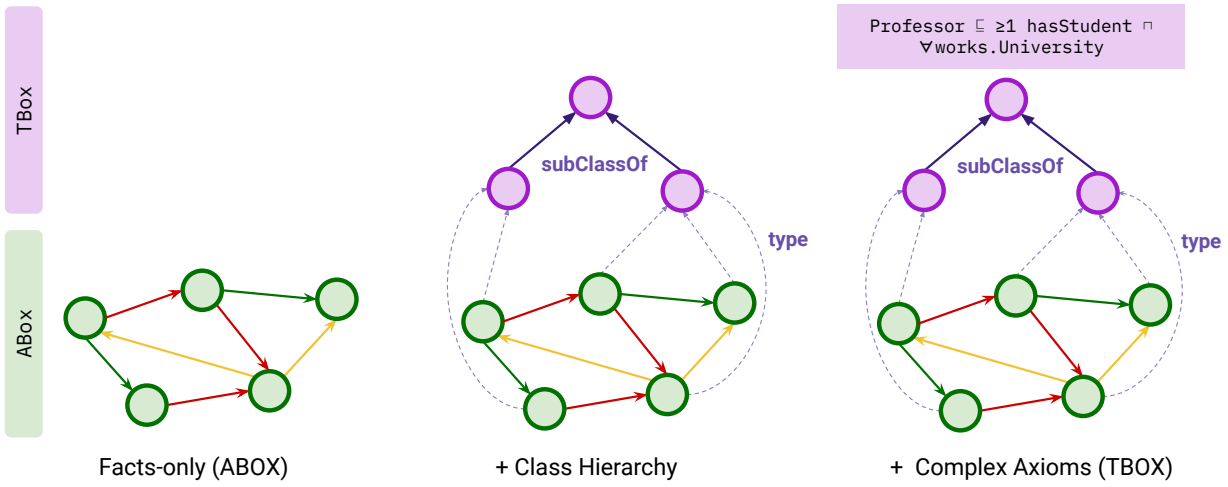

Figure 11: Background Semantics. *Facts-only* are graphs that only have assertions (ABox) and have no higher-level schema. *Class Hierarchy* introduces node types (classes) and hierarchical relationships between classes. Finally, *Complex Axioms* add even more complex logical rules (TBox) governed by certain OWL profiles, *e.g.*, *A professor is someone who has one or more students and works at university.*

might describe, for instance, node types are common in the Labeled Property Graph (LPG) paradigm. RDF graphs, however, might provide an additional layer of logical consistence by employing standards based on *Description Logics* (Baader et al., 2003). As incorporating schema is crucial for designing effective query answering ML models, we introduce the *Background Semantics* (Fig. 11) as the notion of additional schema information available on top of plain facts (statements).

Facts-only. In the simplest case, there is no background schema such that a KG consists of statements (facts) only, that is, a KG follows Definition 2.2, $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$. In terms of description logics, a graph only has *assertions* (ABox). Queries, depending on the Reasoning Domain (Section 4.2), involve only entities, relations, and literals. The original GQE (Hamilton et al., 2018) focused on facts-only graphs and the majority of subsequent query answering approaches (Table 4) operate exclusively on schema-less KGs.

Class Hierarchy. Classes of entities, or node types, are a natural addition to a facts-only graph as a basic schema. Extending Definition 2.2 with a set of types $\mathcal{T}$, a graph $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S}, \mathcal{T})$. Each

entity e might have one or more associated classes $type(e) = \{t_1, \ldots, t_k\} | t \in \mathcal{T}$. Both LPG and RDF graphs support classes albeit in RDF it is possible to specify hierarchical relationships between classes using RDF Schema (RDFS) (Brickley et al., 2014)[5]. Formally, types become explicit nodes in a graph, and statements $\mathcal{S}$ can now have types as subjects or objects, $\mathcal{S} \subset ((\mathcal{E} \cup \mathcal{T}) \times \mathcal{R} \times (\mathcal{E} \cup \mathcal{T}))$, or other statement elements in graphs of other modalities. Practically, edges involving types might be present physically in a KG or be considered as an additional supervised input to a particular model.

To date, we are aware of three query answering approaches (Table 4) that incorporate entity types. Contextual Graph Attention (CGA, Mai et al. (2019)) only uses types for entity embedding initialization and requires each entity to have only one type. The queries are not conditioned by entity types and the answer set still includes entities only. That is, query constants, variables, and the answer set follow $\texttt{Con} \subseteq \mathcal{E}, \texttt{Var} \subseteq \mathcal{E}, \mu \subseteq \mathcal{E}$.

In Type-Aware Message Passing (Temp, Hu et al. (2022)), type embeddings are used to enrich entity and relation representations that are later sent to a downstream query answering model. Each node might have several types. In the inductive scenario (we elaborate on inference scenarios in Section 5.1) with unseen nodes at inference time, type and relation embeddings are learned *invariants* that transfer across training and inference entities. Query-wise, constants, variables, and the answer set are still limited to entities only.

The TBox and ABox Neural Reasoner (TAR, Tang et al. (2022)) incorporates types and their hierarchy to improve predictions over entities as well as introduces a task of predicting types of answer entities, that is, $\mu \subseteq (\mathcal{E} \cup \mathcal{T})$. Constants and variables are still entity-only such that using types as query variables is not allowed. The class hierarchy in TAR is used in three auxiliary losses besides the original entity prediction, that is, *concept retrieval* – prediction of the answer set of types, *subsumption* – predicting which type is a subclass of another type, and *instantiation* – predicting a type for each entity.

A natural next step for the *Class Hierarchy* family of approaches is to incorporate types in queries in the form of constants and variables, $\texttt{Con} \subseteq (\mathcal{E} \cup \mathcal{T}), \texttt{Var} \subseteq (\mathcal{E} \cup \mathcal{T})$.

Complex Axioms. Finally, a schema might contain not just a class hierarchy but a set of more complex axioms involving, for example, a hierarchy of relations, qualified restrictions on relations, or composite classes. Such a complex schema can now be treated as an *ontology* $\mathcal{O}$ that extends the definition of the graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S}, \mathcal{O})$. In Fig. 11, the axiom $\texttt{Professor} \sqsubseteq \geq 1\ hasStudent \sqcap \forall works.\texttt{University}$ describes that *A professor is someone who has one or more students and works at university.* In terms of description logics, a graph has an additional *terminology* component (TBox). The expressiveness of TBox directly affects the complexity of symbolic reasoning engines up to exponential (ExpTime) for most expressive fragments. To alleviate this issue, ontology languages (like Web Ontology Language, OWL) specify less expressive but computationally feasible *profiles*, e.g., OWL 2 introduces three profiles *OWL EL*, *OWL QL*, *OWL RL* (Motik et al., 2009). *OWL-QL*, in turn, is loosely connected to the *DL-Lite*$_{\mathcal{R}}$ ontology language (Artale et al., 2009).

In graph representation learning, incorporating complex ontological axioms is non-trivial even for simple link prediction models (Zhang et al., 2022b). In the query answering literature, the only attempt to include complex axioms is taken by Andresel et al. (2021). In Q2B Onto (O2B), an extension of Query2Box (Q2B, Ren et al. (2020)), the set of considered complex axioms belongs to the *DL-Lite*$_{\mathcal{R}}$ fragment and supports the hierarchy of classes (*subclasses*), the hierarchy of relations (*subproperties*), as well as *range* and *domain* of relations. The model architecture is not directly conditioned on the axioms and remains the original Query2Box. Instead, the axioms affect the graph structure, query sampling, and an auxiliary loss, that is, *query rewriting* mechanisms are used to materialize more answers to original queries as if executed against the complete graph (*deductive closure*) akin to data augmentation. During optimization, an auxiliary regularization loss aims at including a specialized query box q into the more general version of this query q'.

Still, even the expensive procedure of incorporating complex axioms in query sampling in O2B benefits mostly the *deductive* capabilities of query answering, that is, inferring answers that are already implied by the graph $\mathcal{G}$ and ontology $\mathcal{O}$, and does not improve the *generalization* capabilities when missing edges cannot be inferred by ontological axioms. We elaborate on *deductive*, *generalization*, and other setups in Section 7.

[5]RDFS has more expressive means (*e.g.*, a hierarchy of relations) but we leave them to *Complex Axioms*

Another avenue for future work is a better understanding of theoretical expressiveness of Graph Neural Network (GNN) encoders when applied to multi-relational KGs. Initial works on non-relational graphs (Barceló et al., 2020) map the expressiveness to the FOC_2 subset of FOL with two variables and counting quantifiers, and to FOC_B (Luo et al., 2022) for hypergraphs of maximum arity B. In relational graphs, Barceló et al. (2022) quantified the expressiveness of relational GNNs in terms of the *relational Weisfeiler-Leman* (RWL) test proving that RWL is more expressive than classical WL test (Weisfeiler & Leman, 1968) and that common relational GNN architectures like R-GCN (Schlichtkrull et al., 2018) and CompGCN (Vashishth et al., 2020) are bounded by 1-RWL. Using RWL, Huang et al. (2023) derive that the family of GNNs conditioned on the query node, such as Neural Bellman-Ford Networks (Zhu et al., 2021), are bounded by the asymmetric local 2-RWL and expressive as $\text{rGFO}^3_{\text{cnt}}$, restricted guarded first-order logic fragment with three variables and counting. Concurrently, Gao et al. (2023) study KGs as double permutation equivariant structures (to permuting nodes and edge types) and map their expressiveness to universally quantified entity-relation (UQER) Horn clauses. However, it is still an open question if GNNs can capture OWL-like axioms and leverage them as an inductive bias in complex query answering.

Table 4: Complex Query Answering approaches categorized under *Background Semantics* .

Facts-only (ABOX)	+ Class Hierarchy	+ Complex Axioms (TBOX)
GQE (Hamilton et al., 2018), GQE w hash (Wang et al., 2019), TractOR (Friedman & Broeck, 2020), Query2Box (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), EmQL (Sun et al., 2020), Shv (Gebhart et al., 2021), RotatE-Box (Adlakha et al., 2021), MPQE (Daza & Cochez, 2020), BiQE (Kotnis et al., 2021), HyPE (Choudhary et al., 2021b), NewLook (Liu et al., 2021), CQD (Arakelyan et al., 2021), PERM (Choudhary et al., 2021a), ConE (Zhang et al., 2021b), LogicE (Luus et al., 2021), MLPMix (Amayuelas et al., 2022), FuzzQE (Chen et al., 2022b), GNN-QE (Zhu et al., 2022), GNNQ (Pflueger et al., 2022), SMORE (Ren et al., 2022), KGTrans (Liu et al., 2022), LinE (Huang et al., 2022b), Query2Particles (Bai et al., 2022a), FLEX (Lin et al., 2022b), TFLEX (Lin et al., 2022a), NodePiece-QE (Galkin et al., 2022b), ENeSy (Xu et al., 2022), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022), StarQE (Alivanistos et al., 2022), QTO (Bai et al., 2022b), SignalE (Wang et al., 2022a), LMPNN (Wang et al., 2023c), NQE (Luo et al., 2023), Var2Vec (Wang et al., 2023a), $\text{CQD}^{\mathcal{A}}$ (Arakelyan et al., 2023), Query2Geom (Sardina et al., 2023), SQE (Bai et al., 2023)	CGA (Mai et al., 2019), TeMP (Hu et al., 2022), TAR (Tang et al., 2022)	Q2B Onto (Andresel et al., 2021)

5 Modeling

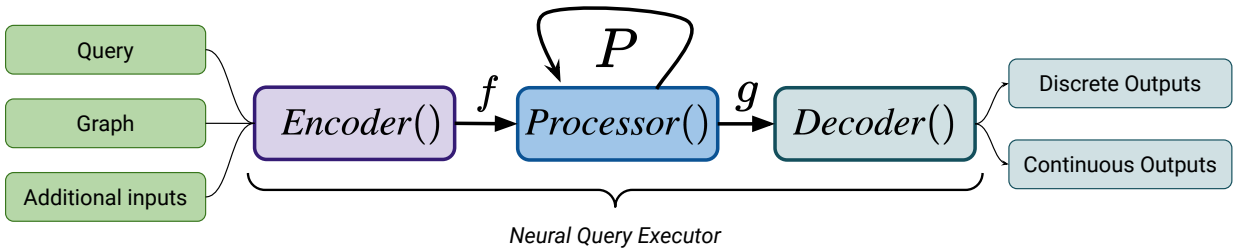

Figure 12: *Neural Query Execution* through the *Encoder-Processor-Decoder* modules. Encoder function f builds representations of inputs (query, target graph, auxiliary data) in the latent space. Processor P executes the query with its logical operators against the graph conditioned on other inputs. Decoder function g builds requested outputs that might be discrete or continuous.

In this section, we discuss the literature from the perspective of *Modeling*. Following the common methodology (Battaglia et al., 2018), we segment the *Modeling* methods through the lens of *Encoder-Processor-Decoder*

modules (illustrated in Fig. 12). (1) The *Encoder* ENC() takes an input query q, target graph $\mathcal{G}$ with its entities and relations, and auxiliary inputs (*e.g.*, node, edge, graph features) to build their representations in the latent space. (2) The *Processor* P leverages the chosen inductive biases to process representations of the query with its logical operators in the latent or symbolic space. (3) The *Decoder* DEC() takes the processed latents and builds desired outputs such as a distribution over discrete entities or regression predictions in case of continuous tasks. Generally, encoder, processor, and decoder can be parameterized with a neural network θ or be non-parametric. Finally, we analyze computational complexity of existing processors.

5.1 Encoder

We start the modeling section with encoders, *i.e.*, how different methods encode and represent entities and relations from the KG. There are three different categories, *Shallow Embedding*, *Transductive Encoder*, and *Inductive Encoder* each representing a different way of producing the neural representation of the entities/relations. Different encoding methods are suitable in different inference setups (details in Section 7.4), and may further require different logical operator methods (details in Section 5.2). Fig. 13 illustrates the three common encoding approaches.

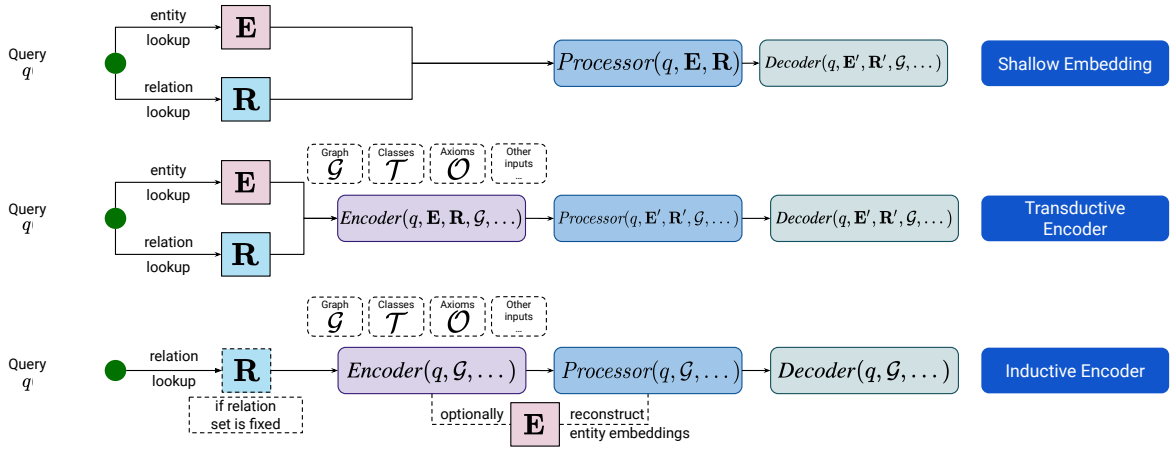

Figure 13: Categorization of *Encoders*. Shallow encoders perform entity and relation embedding lookup and send them to the processor. Transductive encoders additionally enrich the representations with query, graph, classes, or other latents. Inductive encoders do not need learnable entity embeddings.

Table 5: Complex Query Answering approaches categorized under *Encoder*.

Shallow Embedding	Transductive Encoder	Inductive Encoder
GQE (Hamilton et al., 2018), GQE w hash (Wang et al., 2019), CGA (Mai et al., 2019), TractOR (Friedman & Broeck, 2020), Query2Box (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), EmQL (Sun et al., 2020), Shv (Gebhart et al., 2021), Q2B Onto (Andresel et al., 2021), RotatE-Box (Adlakha et al., 2021), HyPE (Choudhary et al., 2021b), NewLook (Liu et al., 2021), CQD (Arakelyan et al., 2021), PERM (Choudhary et al., 2021a), ConE (Zhang et al., 2021b), LogicE (Luus et al., 2021), FuzzQE (Chen et al., 2022b), SMORE (Ren et al., 2022), LinE (Huang et al., 2022b), Query2Particles (Bai et al., 2022a), TAR (Tang et al., 2022), FLEX (Lin et al., 2022b), TFLEX (Lin et al., 2022a), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022), QTO (Bai et al., 2022b), SignalE (Wang et al., 2022a), Var2Vec (Wang et al., 2023a), CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023), Query2Geom (Sardina et al., 2023)	MPQE (Daza & Cochez, 2020), BiQE (Kotnis et al., 2021), KGTrans (Liu et al., 2022), StarQE (Alivanistos et al., 2022), MLPMix (Amayuelas et al., 2022), ENeSy (Xu et al., 2022), LMPNN (Wang et al., 2023c), NQE (Luo et al., 2023), SQE (Bai et al., 2023)	NodePiece-QE (Galkin et al., 2022b), GNN-QE (Zhu et al., 2022), GNNQ (Pflueger et al., 2022), TeMP (Hu et al., 2022)

Shallow Embeddings. The first line of approaches encodes each entity/relation on the graph as a low-dimensional vector, and thus we achieve an entity embedding matrix $\boldsymbol{E}$ and a relation embedding matrix $\boldsymbol{R}$.

The shape of the entity embedding matrix is $|\mathcal{E}| \times d$ ($|\mathcal{R}| \times d$), where d is the dimension of the embedding. Shallow embedding methods assume independence of the representation of all the nodes on the graph. This independence assumption gives the model much freedom, free parameters to learn. Such modeling origins from the KG completion literature, where the idea is to learn the entity and relation embedding matrices by optimizing a pre-defined distance/score function over all edges on the graph, *e.g.*, a triplet fact $\texttt{dist}(\mathbf{e_s}, \mathbf{r}, \mathbf{e_o})$. The majority of query answering literature follows the same paradigm with various different embedding spaces and distance functions to learn the entity and relation embedding matrices. Multiple embedding spaces have been proposed. For example, GQE (Hamilton et al., 2018) and Query2Box (Ren et al., 2020) embed into $\mathbb{R}^d$ (point vector in the Euclidean space); FuzzQE (Chen et al., 2022b) embeds into the space of real numbers in range $[0, 1]$ (fuzzy logic score); BetaE (Ren & Leskovec, 2020) uses Beta distribution, a probabilistic embedding space; ConE Zhang et al. (2021b) on the other hand embeds entities as a point on a unit circle. Each design choice motivates the inductive bias for executing logical operators (as we show in Section 5.2). Some approaches employ shallow entity and relation embeddings already pre-trained on a simple link prediction task and just apply on top of them a query answering decoder with non-parametric logical operators. For example, CQD (Arakelyan et al., 2021), LMPNN (Wang et al., 2023c), Var2Vec (Wang et al., 2023a), and CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023) take pre-trained embeddings in the complex space $\mathbb{C}^d$ and apply non-parametric *t-norms* and *t-conorms* to model intersection and union, respectively. QTO (Bai et al., 2022b) goes even further and fully materializes scores of all possible triples in one $[0, 1]^{|\mathcal{R}| \times |\mathcal{E}| \times |\mathcal{E}|}$ matrix given pre-trained entity and relation embeddings at preprocessing stage.

Despite being the mainstream design choice, the downside of shallow methods is that (1) shallow embeddings do not use any inductive bias and prior knowledge of the entity or its neighboring structure since the parameters of all entities/relations are free parameters learned from scratch; (2) they are not applicable in the inductive inference setting since these methods do not have a representation/embedding for those unseen novel entities by design. One possible solution is to randomly initialize one embedding vector for a novel entity and finetune the embedding vector by sampling queries involving the novel entity (detailed in Section 7.3) However, such a solution requires gradient steps during inference, rendering it not ideal.

Transductive Encoder. Similar to shallow embedding methods, transductive encoder methods learn the same entity embedding matrix $\mathcal{E}$. Besides, they learn an additional encoder $\textsc{Enc}_\theta(q, \boldsymbol{E}, \boldsymbol{R}, \dots)$ (parameterized with θ) on top of the query q, entity and relation embedding matrices (and, optionally, other available inputs). The goal is to apply the encoder to the embeddings of entities in the query q in order to capture dependencies between neighboring entities in the graph. Specifically, the additional encoder may take several rows of the feature matrix as input and further apply transformations. For example, BiQE (Kotnis et al., 2021) and kgTransformer (Liu et al., 2022) linearize a query graph $\mathcal{G}_q$ into a sequence and apply a Transformer (Vaswani et al., 2017) encoder that attends to all other embeddings in the query and obtain the final representation of the `[MASK]` token as the target query. MPQE (Daza & Cochez, 2020) and StarQE (Alivanistos et al., 2022) run a message passing GNN architecture on top of the query graph $\mathcal{G}_q$ to enrich entity and relation embeddings and extract the final node representation as the query embedding. These methods share similar benefit and disadvantage of the shallow embeddings. Namely, there are many free parameters in the method to train. Unlike the shallow embeddings, the additional encoder leverages relational inductive bias between an entity and its neighboring entities or other entities in a query, allowing for a better learned entity representation and generalization capacity. However, since at its core the method is still based on the large look-up matrix of entity embeddings, it still exhibits the same downside that all such methods cannot be directly applied to an inductive setting where we may observe new entities.

Inductive Encoder. In order to address the aforementioned challenges of shallow embeddings and transductive encoders, inductive encoder methods aim to avoid learning an embedding matrix $\boldsymbol{E}$ for a fixed number of entities. Instead, inductive representations are often calculated by leveraging certain *invariances*, that is, the features that remain the same when transferred onto different graphs with new entities at inference time. As we describe in Section 7.4, inductive encoders might employ different invariances albeit the majority of inductive encoders rely on the assumption of the fixed set of relation types $\mathcal{R}$. Formally, following Definition 2.7, given a complex query $\mathcal{Q} = (\mathcal{E}', \mathcal{R}', \mathcal{S}', \bar{\mathcal{S}}')$ composed of entity and relation terms $\mathcal{E}', \mathcal{R}'$ (that, in turn, contain constants `Con` and variables `Var`), relation projections $R(a, b) \in \mathcal{S}$ (and, optionally, in

$\bar{\mathcal{S}}'$), a target graph $\mathcal{G}$ (and, optionally, other inputs), inductive encoders learn a conditional representation function $\textsc{Enc}_\theta(e|\mathcal{E}', \mathcal{R}', \mathcal{G}, \dots)$ for each entity $e \in \mathcal{E}$. Galkin et al. (2022b) devise two families of inductive representations, *i.e.*, (1) inductive *node representations* and (2) inductive *relational structure* representations.

Inductive **node representation** approaches parameterize $\textsc{Enc}_\theta$ as a function of a fixed-size invariant vocabulary. For instance, NodePiece-QE (Galkin et al., 2022b) employs the invariant vocabulary of relation types and parameterizes each entity through the set of incident relations. TeMP (Hu et al., 2022) employs the invariant vocabulary of entity types and class hierarchy and injects their representations into entity representations. Inductive node representation approaches reconstruct embeddings of new entities and can be used as a drop-in replacement of shallow lookup tables paired with any *processor* method, *e.g.*, NodePiece-QE used CQD as the processor while TeMP was probed with GQE, Query2Box, BetaE, and LogicE processors.

Inductive **relational structure** representation methods parameterize $\textsc{Enc}_\theta$ as a function of the relative relational structure that only requires learning of relation embeddings and uses relations as invariants. Such methods often employ various *labeling tricks* (Zhang et al., 2021a) to label constants (anchor entities) of the input query $\mathcal{Q}$ such that after the message passing procedure all other nodes would encode a graph structure relative to starting nodes. In particular, GNN-QE (Zhu et al., 2022) labels anchor nodes with the embedding vector of the queried relations, *e.g.*, for a projection query $(h, r, ?)$ a node h will be initialized with the embedding of relation r, whereas all other nodes are initialized with the zero vector. In this way, GNN-QE learns only relation embeddings $\boldsymbol{R}$ and GNN weights. GNNQ (Pflueger et al., 2022) represents a query with its variables and relations as a hypergraph and learns a relational structure through applying graph convolutions on hyperedges. Hyperedges are parameterized with multi-hot feature vectors of participating relations, so the only learnable parameters are GNN weights.

Still, there exists a set of open problems for inductive models. As the majority of inductive methods rely on learning relation embeddings, they cannot be easily used in setups where at inference time KGs are updated with new, unseen relation types, that is, relations are not invariant. This fact might require exploration of novel invariances and featurization strategies (Huang et al., 2022a; Gao et al., 2023; Chen et al., 2023). Inductive models are more expensive to train in terms of both time and memory than shallow models and cannot yet be easily extended to large-scale graphs. We conjecture that inductive encoders will be in the focus of the future work in CLQA as generalization to unseen entities and graphs at inference time without retraining is crucial for updatability of NGDB. Furthermore, updatability might increase the role of *continual learning* (Thrun, 1995; Ring, 1998) and amplify the negative effects of *catastrophic forgetting* (McCloskey & Cohen, 1989) that have to be addressed by the encoders. Larger inference graphs also present a major *size generalization* issue (Yehudai et al., 2021; Buffelli et al., 2022; Zhou et al., 2022b) when performance of GNNs trained on small graphs decreases when running inference on much larger graphs. The phenomenon has been observed by Galkin et al. (2022b) in the inductive complex query answering setup.

5.2 Processor

Having encoded the query and other available inputs, the *Processor* P executes the query in the latent (or symbolic) space against the input graph. Recall that a query q is defined as $q(\mathcal{E}', \mathcal{R}', \mathcal{S}, \bar{\mathcal{S}})$ where $\mathcal{E}'$ and $\mathcal{R}'$ terms include constants `Con` and variables `Var`, statements in $\mathcal{S}$ and $\bar{\mathcal{S}}$ include relation projections $R(a, b)$, and logical operators *ops* over the variables. We define *Processor* P as a collection of modules that perform relation projections $R(a, b)$ given constants `Con` and logical operators $ops \subseteq \{\wedge, \vee, \neg, \dots\}$ over variables `Var` (we elaborate on the logical operators in Section 6.1). Depending on the chosen inductive biases and parameterization strategies behind those modules, we categorize *Processors* into *Neural* and *Neuro-Symbolic* (Table 6a). Furthermore, we break down the Neuro-Symbolic processors into *Geometric*, *Probabilistic*, and *Fuzzy Logic* (Table 6b). Note that in this section we omit pure encoder approaches like TeMP (Hu et al., 2022) and NodePiece-QE (Galkin et al., 2022b) that can be paired with any neural or neuro-symbolic processor. To describe processor models more formally, we denote $\mathbf{e}$ as an entity vector, $\mathbf{r}$ as a relation vector, and $\mathbf{q}$ as the query embedding that is often a function of $\mathbf{e}$ and $\mathbf{r}$. We use $\mathcal{G}_q$ as the query graph.

Neural Processors. Neural processors execute relation projections and logical operators directly in the latent space $\mathbb{R}^d$ parameterizing them with neural networks. To date, most existing purely neural approaches operate exclusively on the query graph $\mathcal{G}_q$ only executing operators within a single query and do not condition

Table 6: Categorization of Query Processors. Table 6a provides a general view on Neural, Symbolic, Neuro-Symbolic methods as well as encoders that can be paired with any processor. Table 6b further breaks down neuro-symbolic processors.

(a) Complex Query Answering approaches categorized under the *Processor* type.

Any Processor	Neural	Neuro-Symbolic
TeMP (Hu et al., 2022), NodePiece-QE (Galkin et al., 2022b)	GQE (Hamilton et al., 2018), GQE w/ hashing (Wang et al., 2019), CGA (Mai et al., 2019), BiQE (Kotnis et al., 2021), MPQE (Daza & Cochez, 2020), StarQE (Alivanistos et al., 2022), MLPMix (Amayuelas et al., 2022), Query2Particles (Bai et al., 2022a), KGTrans (Liu et al., 2022), RotatE-m, DistMult-m, ComplEx-m (Ren et al., 2022), GNNQ (Pflueger et al., 2022), SignalE (Wang et al., 2022a), LMPNN (Wang et al., 2023c), SQE (Bai et al., 2023)	Table 6b

(b) Neuro-symbolic Processors

Geometric [Neuro-Symbolic]	Probabilistic [Neuro-Symbolic]	Fuzzy Logic [Neuro-Symbolic]
Query2Box (Ren et al., 2020), Query2Onto (Andresel et al., 2021), RotatE-Box (Adlakha et al., 2021), NewLook (Liu et al., 2021), Knowledge Sheaves (Gebhart et al., 2021), HypE (Choudhary et al., 2021b), ConE (Zhang et al., 2021b), Query2Geom (Sardina et al., 2023)	BetaE (Ren & Leskovec, 2020), PERM (Choudhary et al., 2021a), LinE (Huang et al., 2022b), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022)	EmQL (Sun et al., 2020), TractOR (Friedman & Broeck, 2020), CQD (Arakelyan et al., 2021), LogicE (Luus et al., 2021), FuzzQE (Chen et al., 2022b), TAR (Tang et al., 2022), FLEX (Lin et al., 2022b), TFLEX (Lin et al., 2022a), GNN-QE (Zhu et al., 2022), ENeSy (Xu et al., 2022), QTO (Bai et al., 2022b), NQE (Luo et al., 2023), Var2Vec (Wang et al., 2023a), CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023)

the execution process on the full underlying graph structure $\mathcal{G}$. Since the query processing is performed in the latent space with neural networks where Union ($\vee$) and Negation ($\neg$) are not well-defined, the majority of neural processors implement only relation projection ($R(a, b)$) and intersection ($\wedge$) operators. We aggregate the characteristics of neural processors as to their embedding space, the way of executing relation projection, logical operators, and the final decoding distance function in Table 7. We illustrate the difference between two families of neural processors (sequential execution and joint query encoding) in Fig. 14.

The original GQE (Hamilton et al., 2018) is the first example of the neural processor. That is, queries $\mathbf{q}$, entities $\mathbf{e}$, and relations $\mathbf{r}$ are vectors in $\mathbb{R}^d$. Query embedding starts with embeddings of constants $\mathcal{C}$ (anchor nodes $\mathbf{e}$) and they get progressively refined through relation projection and intersection, *i.e.*, it is common to assume that query embedding at the initial step 0 is equivalent to embedding(s) of anchor node(s), $\mathbf{q}^{(0)} = \mathbf{e}$. Relation projection is executed in the latent space with the translation function $\mathbf{q} + \mathbf{r}$, and intersection is modeled with the permutation-invariant DeepSet (Zaheer et al., 2017) neural network. Several follow-up works improved GQE to work with hashed binary vectors $\{+1, -1\}^d$ (Wang et al., 2019) or replaced DeepSet with self-attention and translation-based projection to a matrix-vector product (Mai et al., 2019). Recently, Ren et al. (2022) proposed DistMult-m, ComplEx-m, and RotatE-m, extensions of simple link prediction models for complex queries that, inspired by GQE, perform relation projection by the respective composition function and model the intersection operator with DeepSet and, optionally, L2 norm.

The other line of works apply neural encoders to whole query graphs $\mathcal{G}_q$ without explicit execution of logical operators. Depending on the query graph representation, such encoders are often GNNs, Transformers, or MLPs. It is assumed that neural encoders can implicitly capture logical operators in the latent space during optimization. For instance, MPQE (Daza & Cochez, 2020), StarQE (Alivanistos et al., 2022), and LMPNN (Wang et al., 2023c) represent queries as relational graphs (optionally, hyper-relational graphs for StarQE) where each edge is a relation projection and intersection is modeled as two incoming projections to the same variable node. All constants $\mathcal{C}$ and known relation types are initialized from the respective embedding matrices. All variable nodes in all query graphs are initialized with the same learnable `[VAR]` feature vector while all target nodes are initialized with the same `[TAR]` vector. Then, the query graph is

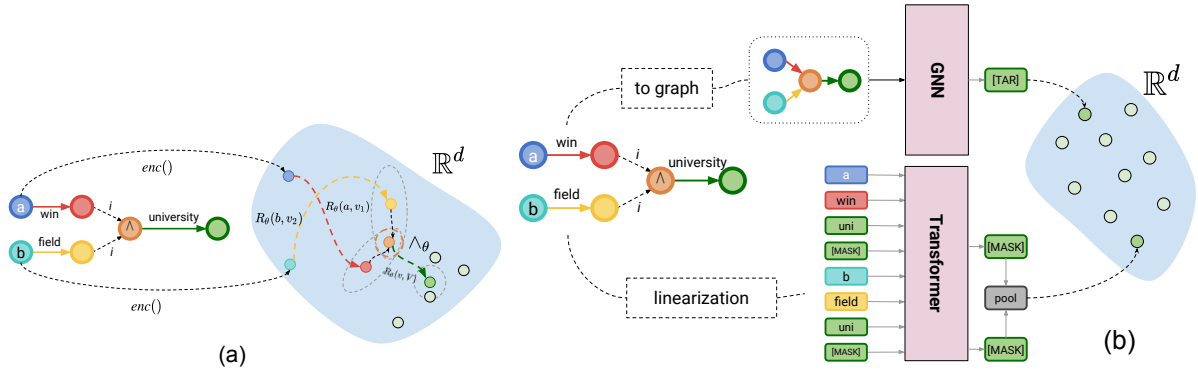

Figure 14: Neural Processors. (a) Relation projections $R_\theta(a, b)$ and logical operators (non-parametric or parameterized with θ) are executed sequentially in the latent space; (b) a query is encoded to a graph or linearized to a sequence and passed through the encoder (GNN or Transformer, respectively). A pooled representation denotes the query embedding.

passed through a GNN encoder (R-GCN (Schlichtkrull et al., 2018) for MPQE, StarE (Galkin et al., 2020) for StarQE, GIN (Xu et al., 2019) for LMPNN), and the final state of the `[TAR]` target node is considered the final query embedding ready for decoding. A recent LMPNN extends query graph encoding with an additional edge feature indicating whether a given projection $R(a, b)$ has a negation or not and derives a closed-form solution for the merged projection and negation operator for the ComplEx composition function. A different approach is taken by GNNQ (Pflueger et al., 2022) that frames query answering as a subgraph classification task. That is, an input query is not directly executed over a given graph $\mathcal{G}$, but, instead, the task is to classify whether a given precomputed subgraph $\mathcal{G}' \subset \mathcal{G}$ satisfies a given conjunctive query. For that, GNNQ first augments the graph with Datalog-derived triples and converts the subgraph to a hypergraph where only hyperedges are parameterized with learnable vectors. On the one hand, this strategy allows GNNQ to be inductive and not learn entity embeddings. On the other hand, GNNQ is limited to conjunctive queries only and extensions to union and negation queries are not defined.

A more exotic approach by Gebhart et al. (2021) is based on the sheaf theory and algebraic topology (Hansen & Ghrist, 2019). There, a graph is represented as a cellular sheaf and conjunctive queries are modeled as chains of relations (0-cochains). A sheaf is induced over the query graph and relevant answers should be consistent with the induced sheaf and entity embeddings. The optimization problem is a harmonic extension of a 0-cochain using *sheaf Laplacian* and *Schur complement* of the sheaf Laplacian. Conceptually, this approach merges execution of projection and intersection operators as functions over topological structures.

Considering Transformer encoders, BiQE (Kotnis et al., 2021), kgTransformer (Liu et al., 2022), and SQE (Bai et al., 2023) linearize a conjunctive query graph into a sequence of relational paths composed of entity constants $\mathcal{C}$ and relation tokens. The order of tokens in paths and intersections of paths are marked with positional encodings. The target node (present in many paths) is marked with the `[MASK]` token (optionally, kgTransformer also annotates existentially quantified variables with `[MASK]`). SQE does not model variables explicitly but instead relies on auxiliary *bracket* tokens that separate branches of the computation graph. Passing the sequence through the Transformer encoder, the final query embedding is the aggregated representation of the target node. BiQE only supports conjunctive queries while kgTransformer converts queries with unions to the Disjunctive Normal Form (DNF) with post-processing of score distributions (we elaborate on query rewritings and normal forms in Section 6.1). SQE explicitly includes all operator tokens into the linearized sequence and thus supports negations.

Finally, MLPMix (Amayuelas et al., 2022) sequentially executes operations of the query where projection, intersection, and negation operators are modeled as separate learnable MLPs. Union queries are converted to DNF such that they can be answered with projection and intersection operators with the final post-processing of scores as a union operator. Similarly, Query2Particles (Bai et al., 2022a) represents each query

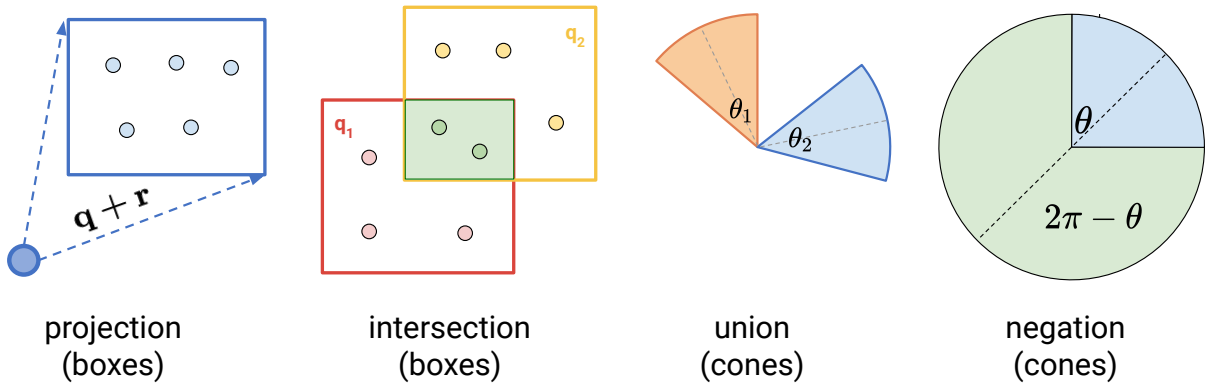

Figure 15: Geometric Processors and their inductive biases.

as a set of vectors in the embedding space and models projection, intersection, and negation operators as attention over the set of particles followed by an MLP. Union is a concatenation of query particles.

Table 7: Neural Processors. Most methods implement only Relation Projection and Intersection operators. The top part of models execute a query sequentially, the bottom part encode the whole query graph $\mathcal{G}_q$.

Model	Embedding Space	Relation Projection	Intersection	Union	Negation	Distance
GQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\mathbf{q} + \mathbf{r}$	$\texttt{DeepSet}(\{\mathbf{q_i}\})$	-	-	$\lVert \mathbf{q} - \mathbf{e} \rVert$
GQE+hashing	$\mathbf{q}, \mathbf{e} \in \{\pm 1\}^d, \mathbf{r} \in \mathbb{R}^d$	$\texttt{sgn}(\mathbf{q} + \mathbf{r})$	$\texttt{sgn}(\texttt{DeepSet}(\{\mathbf{q_i}\}))$	-	-	$-cos(\mathbf{q}, \mathbf{e})$
CGA	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\mathbf{W}_r \mathbf{q}$	$\texttt{SelfAttn}(\{\mathbf{q_i}\})$	-	-	$-cos(\mathbf{q}, \mathbf{e})$
RotatE-m	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$	$\mathbf{q} \circ \mathbf{r}$	$\texttt{DeepSet}(\{\mathbf{q_i}\})$	-	-	$\lVert \mathbf{q} - \mathbf{r} \rVert$
DisMult-m	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\texttt{L2Norm}(\mathbf{q} \circ \mathbf{r})$	$\texttt{L2Norm}(\texttt{DeepSet}(\{\mathbf{q_i}\}))$	-	-	$-\langle \mathbf{q}, \mathbf{e} \rangle$
ComplEx-m	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$	$\texttt{L2Norm}(\mathbf{q} \circ \mathbf{r})$	$\texttt{L2Norm}(\texttt{DeepSet}(\{\mathbf{q_i}\}))$	-	-	$-\text{Re}(\langle \mathbf{q}, \bar{\mathbf{e}} \rangle)$
Query2Particles	$\mathbf{q}, \mathbf{e} \in \mathbb{R}^{d \times K}, \mathbf{r} \in \mathbb{R}^d$	$f(\mathbf{q}, \mathbf{r})$ f is neural gates	$\texttt{MLP}(\texttt{Attn}([\mathbf{q_1}, \mathbf{q_2}]))$	$[\mathbf{q}^1, \ldots, \mathbf{q}^N]$	$\texttt{MLP}(\texttt{Attn}(\mathbf{q}))$	$-\max_k \langle \mathbf{q}^k, \mathbf{e} \rangle$
SignalE	$\mathbf{q}, \mathbf{e}, \mathbf{r} = [\mathbf{z}, \mathbf{v}]$ $\mathbf{z} \in \mathbb{C}^d, \mathbf{v} = \texttt{IDFT}(\mathbf{z}) \in \mathbb{R}^d$	$\mathbf{z}_q \circ \mathbf{z}_r$	$\texttt{SelfAttn}(\{\mathbf{z}_{q_i}\})$	DNF	$(1 - \mathbf{z}_q^{\text{amp}}, \mathbf{z}_q^{\text{phase}})$	$\beta \lVert \mathbf{v}_q - \mathbf{v}_e \rVert_2 + \lVert \mathbf{z}_q - \mathbf{z}_e \rVert_2$
MLPMix	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\texttt{MLP}_{\texttt{mix}}(\mathbf{q}, \mathbf{r})$	$\texttt{MLP}_{\texttt{mix}}(\mathbf{q_1}, \mathbf{q_2})$	DNF	$\texttt{MLP}(\mathbf{q})$	$\lVert \mathbf{q} - \mathbf{e} \rVert$
MPQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\mathbf{W}_r \mathbf{q}$	$\texttt{RGCN}(\mathbf{q}, \mathcal{G}_q)$	-	-	$cos(\mathbf{q}, \mathbf{e})$
StarQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$f(\mathbf{q}, g(\mathbf{r}, \mathbf{h}_{\text{quals}}))$	$\texttt{StarE}(\mathbf{q}, \mathcal{G}_q)$	-	-	$\texttt{dot}(\mathbf{q}, \mathbf{e})$
GNNQ	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^d$	Hypergraph	$\texttt{RGCN}(\mathcal{G}')$	-	-	BCE
LMPNN	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$	$\rho(\mathbf{q}, \mathbf{r}, \text{dir}, \text{neg})$	$\texttt{GIN}(\mathbf{q}, \mathcal{G}_q)$	DNF	$\rho(\mathbf{q}, \mathbf{r}, \text{dir}, \text{neg})$	$cos(\mathbf{q}, \mathbf{e})$
BiQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	Transformer$(\mathbf{q}, \text{linearized } \mathcal{G}_q)$		-	-	CE
kgTransformer	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	Transformer$(\mathbf{q}, \text{linearized } \mathcal{G}_q)$		DNF	-	$\texttt{dot}(\mathbf{q}, \mathbf{e})$
SQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	LSTM / Transformer$(\mathbf{q}, \text{linearized } \mathcal{G}_q \text{ with operator tokens})$				$\texttt{dot}(\mathbf{q}, \mathbf{e})$
Sheaves	$\mathbf{q}, \mathbf{e} \in \mathbb{R}^d, \mathcal{F}_r \in \mathbb{R}^{d \times d}$	$f(\mathbf{q}, \mathcal{G}_q \text{ as cochain})$		-	-	$\lVert \mathcal{F}_r \mathbf{q} - \mathcal{F}_r \mathbf{e} \rVert$

Neuro-Symbolic Processors. In contrast to purely neural and symbolic models, we define *neuro-symbolic* processors as those who (1) explicitly design logic modules (or neural logical operators) that simulate the real logic/set operations, or rely on various kinds of fuzzy logic formulations to provide a probabilistic view of the query execution process, and (2) execute relation traversal in the latent space. The key difference between neuro-symbolic processors and the previous two is that neuro-symbolic processors explicitly model the logical operations with strong inductive bias so that the processing / execution is better aligned with the symbolic operation (*e.g.*, by imposing restrictions on the embedding space) and more interpretable. We further segment these methods into the following categories.

Geometric Processors. Geometric processors design an entity/query embedding space with different geometric intuitions and further customize neuro-symbolic operators that directly simulate their logical counterparts with similar properties (as illustrated in Fig. 15). We aggregate the characteristics of geometric models as to their embedding space and inductive biases for logical operators in Table 8.

Query2Box (Ren et al., 2020) embeds queries $\mathbf{q}$ as hyper-rectangles (high-dimensional boxes) in the Euclidean space. To achieve that, entities $\mathbf{e}$ and relations $\mathbf{r}$ are embedded as points in the Euclidean space where each

Table 8: Geometric Neuro-Symbolic Processors. $\odot$ denotes element-wise multiplication, $\oplus_c$ denotes Möbius addition, $\odot_c$ denotes Möbius scalar product. `DS` denotes DeepSets neural network. NewLook only partially implements negation as the difference operator.

Model	Embedding Space	Relation Projection	Intersection	Union	Negation	Distance
Query2Box Query2Onto	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^{2d}$, $\mathbf{e} \in \mathbb{R}^d$	$\mathbf{q} + \mathbf{r}$	$\mathbf{q}_c = \texttt{Attn}(\{\mathbf{q_c^i}\})$ $\mathbf{q_o} = \min(\{\mathbf{q_o^i}\}) \odot \sigma(\texttt{DS}(\{\mathbf{q_o^i}\}))$	DNF	-	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$
Query2Geom	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^{2d}$, $\mathbf{e} \in \mathbb{R}^d$	$\mathbf{q} + \mathbf{r}$	$\mathbf{q}_c = \frac{1}{2}((\mathbf{q_c^i} + \mathbf{q_o^i}) + (\mathbf{q_c^j} - \mathbf{q_o^j}))$ $\mathbf{q}_o = \mathbf{q_c} - (\mathbf{q_c^i} - \mathbf{q_o^i})$	DNF	-	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$
RotatE-Box	$\mathbf{q}, \mathbf{r} \in \mathbb{C}^{2d}, \mathbf{e} \in \mathbb{C}^d$	$(\mathbf{q_c} \circ \mathbf{r_c}, \mathbf{q_o} + \mathbf{r_o})$	-	DNF / $\texttt{DS}(\{\mathbf{q}^i\})$	-	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$
NewLook	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^{2d}, \mathbf{e} \in \mathbb{R}^d$ $\mathbf{x} \in \mathcal{T}^{\lvert\mathcal{R}\rvert \times \lvert\mathcal{E}\rvert \times \lvert\mathcal{E}\rvert}$	$\texttt{MLP}[\texttt{MLP}(\mathbf{q_c} + \mathbf{r_c}) \parallel$ $\texttt{MLP}(\mathbf{r_o}) \parallel \mathbf{x}_t]$	$\mathbf{q_c} = \texttt{Attn}(\{\mathbf{q_c^i}, \mathbf{x^i}\})$ $\mathbf{q_o} = \min(\{\mathbf{q_o^i}\}) \odot \sigma(\texttt{DS}(\{\mathbf{q_o^i}\}))$	DNF	$\texttt{Attn}(\{\mathbf{q_c^i}\})$ $\texttt{Attn}(\{\mathbf{q_o^i}, \mathbf{x^i}\})$	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$
HypE	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^{2d}$, $\mathbf{e} \in \mathbb{R}^d$	$\mathbf{q} \oplus_c \mathbf{r}$	$\mathbf{q_c} = \texttt{Attn}(\{\mathbf{q_c^i}\})$ $\mathbf{q_o} = \min(\{\mathbf{q_o^i}\}) \odot_c \sigma(\texttt{DS}(\{\mathbf{q_o^i}\}))$	DNF	-	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$
ConE	$\mathbf{q}, \mathbf{r} = (\boldsymbol{\theta}_{\mathrm{ax}}, \boldsymbol{\theta}_{\mathrm{ap}})$ $\mathbf{e} = (\boldsymbol{\theta}_{\mathrm{ax}}, \mathbf{0})$ $\boldsymbol{\theta}_{\mathrm{ax}} \in [-\pi, \pi)^d$ $\boldsymbol{\theta}_{\mathrm{ap}} \in [0, 2\pi]^d$	$g(\texttt{MLP}(\mathbf{q} + \mathbf{r}))$ g gates $\boldsymbol{\theta}_{\mathrm{ax}}$ and $\boldsymbol{\theta}_{\mathrm{ap}}$	$\boldsymbol{\theta}_{\mathrm{ax}} = \texttt{SemanticAvg}(\mathbf{q}_1, \ldots, \mathbf{q}_n)$ $\boldsymbol{\theta}_{\mathrm{ap}} = \texttt{CardMin}(\mathbf{q}_1, \ldots, \mathbf{q}_n)$	DNF/DM	$\boldsymbol{\theta}_{\mathrm{ax}} = \boldsymbol{\theta}_{\mathrm{ax}} \pm \pi$ $\boldsymbol{\theta}_{\mathrm{ap}} = 2\pi - \boldsymbol{\theta}_{\mathrm{ap}}$	$\mathrm{d_{out}} + \alpha \mathrm{d_{in}}$

relation has an additional learnable offset vector $\mathbf{r_o}$ (entities' offsets are zeros). The projection operator is modeled as an element-wise summation $\mathbf{q} + \mathbf{r}$ of centers and offsets of the query and relation, that is, the initial box is obtained by projecting the original anchor node embedding $\mathbf{e}$ (with zero offset) with the relation embedding $\mathbf{r}$ and relation offset $\mathbf{r_o}$. Accordingly, an attention-based neuro-intersection operator is designed to simulate the set intersection of the query boxes in the Euclidean space. The operator is closed, permutation invariant and aligns well with the intuition that the size of the intersected set is smaller than that of all input sets. The union operator is achieved via DNF, that is, union is the final step of concatenating results of operand boxes. Several works extend Query2Box, *i.e.*, Query2Onto (Andresel et al., 2021) attempts to model complex ontological axioms by materializing entailed triples and enforcing hierarchical relationship using inclusion of the box embeddings; RotatE-Box (Adlakha et al., 2021) designs an additional rotation-based Kleene plus (+) operator denoting relational paths (we elaborate on the Kleene plus operator in Section 6.1); NewLook (Liu et al., 2021) adds symbolic lookup from the adjacency tensor[6] to the operators, modifies projection with MLPs and models the *difference* operator as attention over centers and offsets (note that the difference operator is a particular case of the *2in* negation query, we elaborate on that in Section 6.1); Query2Geom (Sardina et al., 2023) replaces an attention-based intersection operator with a simple non-parametric closed-form geometric intersection of boxes.

HypE (Choudhary et al., 2021b) extends the idea of Query2Box and embeds a query as a hyperboloid (two parallel pairs of arc-aligned horocycles) in a Poincaré hyperball to better capture the hierarchical information. A similar attention-based neuro-intersection operator is designed for the hyperboloid embeddings with the goal to shrink the limits with DeepSets.

ConE (Zhang et al., 2021b), on the other hand, embeds queries on the surface of a set of unit circles. Each query is represented as a cone section and the benefit is that in most cases the intersection of cones is still a cone, and the negation/complement of a cone is also a cone thanks to the angular space bounded by 2π. Based on this intuition, they design geometric neuro-intersection and negation operators.

To sum up, the geometric-based processors are often designed with a strong geometric prior such that properties of the logical/set operations can be better simulated or satisfied.

Probabilistic Processors. Instead of a geometric embedding space, probabilistic processors aim to model the query and the logic/set operations in a probabilistic space. Such methods are similar to the geometric based methods in that they also require a probabilistic space on which one can design a neuro-symbolic logic/set operator that aligns well with the real one. Some examples of implementing logical operators in a probabilistic space are illustrated in Fig. 16. The aggregated characteristics are presented in Table 9.

[6]Incorrect implementation led to the major test set leakage and incorrect reported results.

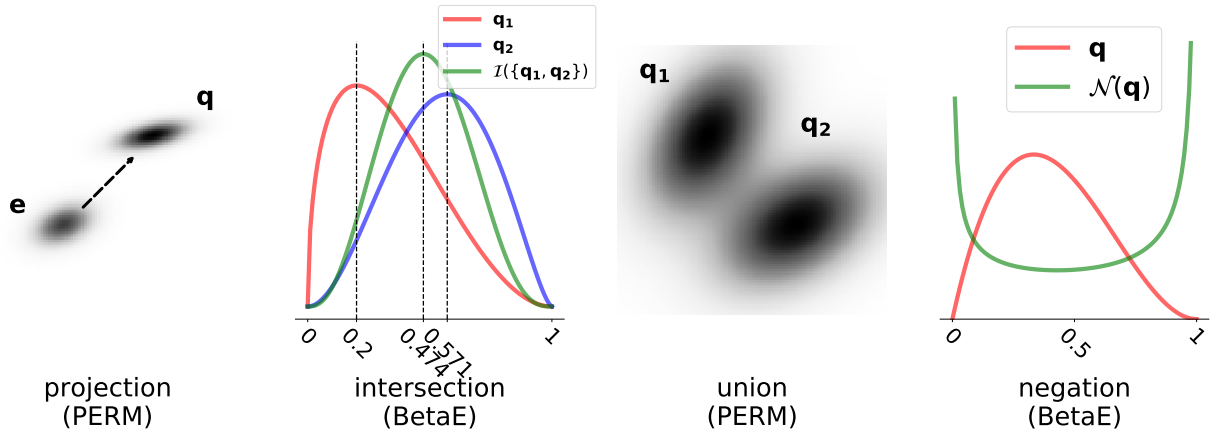

Figure 16: Probabilistic Processors and their inductive biases.

BetaE (Ren & Leskovec, 2020) builds upon the Beta distribution and embeds entities/queries as high-dimensional Beta distribution with learnable parameters. The benefit is that one can design a parameterized neuro-intersection operator over two Beta embeddings where the output is still a Beta embedding with more concentrated density function. A neuro-negation operator can be designed by simply taking the reciprocal of the parameters in order to flip the density. PERM (Choudhary et al., 2021a) looks at the Gaussian distribution space and embeds queries as a multivariate Gaussian distribution. Since the product of Gaussian probability density functions (PDFs) is still a Gaussian PDF, the neuro-intersection operator accordingly calculates the parameters of the Gaussian embedding of the intersected set. NMP-QEM (Long et al., 2022) develops this idea further and represents a query as a mixture of Gaussians where logical operators are modeled with MLP or attention over distribution parameters. LinE (Huang et al., 2022b) transforms the Beta distribution into a discrete sequence of values. A similar neuro-negation operator is introduced by taking the reciprocal as BetaE while designing a new neuro-intersection/union operator by taking element-wise min/max. GammaE (Yang et al., 2022a) replaces Beta distribution with Gamma distribution as entity and query embedding space. Parameterizing logical operators with operations over mixtures of Gamma distributions, union and negation become closed, do not need DNF or DM transformations, and can be executed sequentially along the query computation graph. Overall, probabilistic processors are similar to geometric processors since they are all inspired by certain properties of the probability and geometry used for embeddings and customize neuro-logic operators.

Table 9: Probabilistic Neuro-Symbolic Processors.

Model	Embedding Space	Relation Projection	Intersection	Union	Negation	Distance
BetaE	$\mathbf{q}, \mathbf{e} \in \mathbb{R}^{2d}$	$\texttt{MLP}_r(\mathbf{q})$	$\mathbf{q} = [(\sum w_i\alpha_i, \sum w_i\beta_i)]$	DNF/DM	$\frac{1}{\mathbf{q}}$	$\texttt{KL}(\mathbf{e};\mathbf{q})$
PERM	$\mathbf{q}, \mathbf{e}, \mathbf{r} = \mathcal{N}(\boldsymbol{\mu}, \Sigma)$	$\mathcal{N}(\boldsymbol{\mu}_q + \boldsymbol{\mu}_r, (\Sigma_q^{-1} + \Sigma_r^{-1})^{-1})$	$\Sigma_q^{-1} = \Sigma_{q_1}^{-1} + \Sigma_{q_2}^{-1}$ $\boldsymbol{\mu}_q = \Sigma_q(\Sigma_{q_2}^{-1}\boldsymbol{\mu}_1 + \Sigma_{q_1}^{-1}\boldsymbol{\mu}_2)$	$\texttt{Attn}(\{\mathbf{q}_i\})$	-	$(\boldsymbol{\mu}_q - \boldsymbol{\mu}_e)^T\Sigma_q^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_e)$
LinE	$\mathbf{q}, \mathbf{e} \in \mathbb{R}^{k\times h}$	$\texttt{MLP}_r(\mathbf{q})$	$\min\{q_1, q_2\}$	$\max\{q_1, q_2\}$	$[\frac{1}{p_1^q}, \ldots, \frac{1}{p_k^q}]^h$	$\lVert\mathbf{q} - \mathbf{e}\rVert_2^2$
GammaE	$\mathbf{q}, \mathbf{e} \in \mathbb{R}^{2d}$	$\texttt{MLP}_r(\mathbf{q})$	$\mathbf{q} = [(\sum w_i\alpha_i, \sum w_i\beta_i)]$	$\texttt{Attn}(\{\mathbf{q}_i\})$	$[\frac{1}{\alpha}, \beta]$	$\texttt{KL}(\mathbf{e};\mathbf{q})$
NMP-QEM	$\mathbf{e} \in \mathbb{R}^d$ $\mathbf{q} = \sum_{i=1}^{K} \omega_i\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$	$\texttt{MLP}_r(\mathbf{q})$	$\texttt{Attn}(\{\mathbf{q}_i\})$	DNF	$\texttt{MLP}(\mathbf{q})$	$\lVert\mathbf{e} - \sum_{i=1}^{K} \omega_i^q\boldsymbol{\mu}_i^q\rVert$

Fuzzy-Logic Processors. Unlike the methods above, fuzzy-logic processors directly model all logical operations using existing fuzzy logic theory (Klement et al., 2013; van Krieken et al., 2022) where intersection can be expressed via *t-norms* and union via corresponding *t-conorms* (Section 2.6). In such a way, fuzzy-logic processors avoid the need to manually design or learn neural logical operators as in the previous two processors but rather directly use established fuzzy operators commonly expressed as differentiable, element-wise algebraic operators over vectors (Fig. 17). While intersection, union, and negation are non-parametric, the projection operator might still be parameterized with a neural network. The aggregated characteristics of fuzzy-logic processors are presented in Table 10. Generally, fuzzy processors aim to combine execution

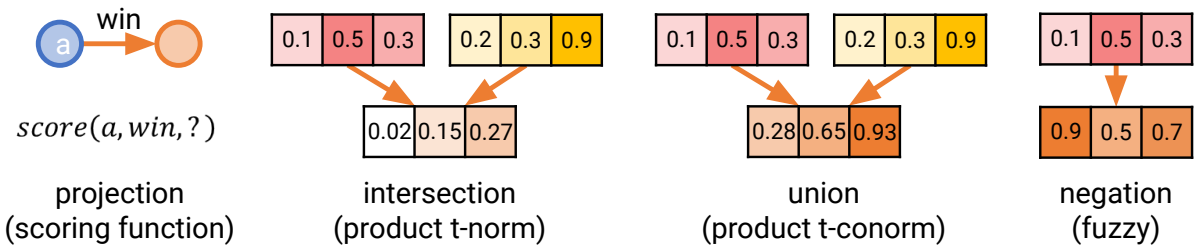

Figure 17: Fuzzy-Logic Processors and fuzzy logical operators.

in *embedding space* (vectors) with *entity space* (symbols). The described methods are different in designing such a combination.

One of the first fuzzy processors is EmQL (Sun et al., 2020) that imbues entity and relation embeddings with a count-min sketch (Cormode & Muthukrishnan, 2005). There, projection, intersection, and union are performed both in the embedding space and in the symbolic sketch space, *e.g.*, intersection is modeled as an element-wise multiplication and union is an element-wise summation of two sketches. CQD (Arakelyan et al., 2021) scores each atomic formula in a query with a pretrained neural link predictor and uses t-norms and t-conorms to compute the final score of a query directly in the *embedding space*. CQD does not train any neural logical operators and only requires pretraining the entity and relation embeddings with one-hop links such that the projection operator is equivalent to *top-k* results of the chosen scoring function, *e.g.*, ComplEx (Lacroix et al., 2018). The idea was then extended in several directions: Query Tree Optimization (QTO) (Bai et al., 2022b) added a look-up from the materialized tensor of scores of all possible triples $\mathbf{M} \in [0,1]^{|\mathcal{R}|\times|\mathcal{E}|\times|\mathcal{E}|}$ to the relation projection step; CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023) and Var2Vec (Wang et al., 2023a) learn an additional linear transformation of the entity-relation concatenation $\mathbf{W}[\mathbf{q}, \mathbf{r}]$.

LogicE (Luus et al., 2021) designs logic embeddings for each entity with a list of *lower bound – upper bound* pairs in range $[0, 1]$, which can be interpreted as a uniform distribution between the lower and upper bound. LogicE executes negation and conjunction with continuous t-norms over the lower and upper bounds. FuzzQE (Chen et al., 2022b) and TAR (Tang et al., 2022) embed query to a high-dimensional fuzzy space $[0, 1]^d$ and similarly use Gödel t-norm and Łukasiewicz t-norm to model disjunction, conjunction and negation. FuzzQE models relation projection as a relation-specific MLP whereas TAR uses a geometric translation (element-wise sum). FLEX (Lin et al., 2022b) and TFLEX (Lin et al., 2022a) embed a query as a mixture of feature and logic embedding. For the logic part, both methods use the real logical operations in vector logic (Mizraji, 2008). TFLEX adds a temporal module conditioning logical operators on the time embedding.

GNN-QE (Zhu et al., 2022) models the likelihood of all entities for each relation projection step with a graph neural network NBFNet (Zhu et al., 2021). It further adopts product logic to directly model the set operations (intersection, union, and negation) over the fuzzy set obtained after a relation projection. GNN-QE employs a node labeling technique where a starting node is initialized with the relation vector (while other nodes are initialized with zeros). This allows GNN-QE to be inductive and not rely on trainable entity embeddings. ENeSy (Xu et al., 2022), on the other hand, maintains both vector and symbolic representations for queries, entities, and relations (where symbolic relations are encoded into $\mathbf{M}_r$ sparse adjacency matrices). Logical operators are executed first in the neural space, *e.g.*, relation projection is RotatE composition function (Sun et al., 2019), and then get intertwined with symbolic representations. Logical operators in the symbolic space employ a generalized version of the product logic and corresponding t-(co)norms.

In summary, fuzzy-logic processors directly rely on established fuzzy logic formalisms to perform all the logical operations in the query and avoid manually designing and learning neural operators in (possibly) unbounded embedding space. The fuzzy logic space is continuous but bounded within $[0, 1]$ – this is both the advantage and weakness of such processors. The bounded space is beneficial for closed logical operators as their output values still belong to the same bounded space. On the other hand, most of the known t-norms (and corresponding t-conorms) still lead to vanishing gradients and only the Product logic norms are stable (van Krieken et al., 2022; Badreddine et al., 2022). Another caveat is designing an effective and

Table 10: Fuzzy Neuro-Symbolic Processors.

Model	Embedding Space	Relation Projection	Intersection	Union	Negation	Distance
EmQL	$\mathbf{e}, \mathbf{r} \in \mathbb{R}^d, \mathbf{q} \in \mathbb{R}^{3d}$ $\mathbf{b}$ count-min sketch	$\texttt{MIPS}(\mathbf{q}, [\mathbf{r}, \mathbf{e}_h, \mathbf{e}_t])$	$(\mathbf{q_1} + \mathbf{q_2})/2$ $\mathbf{b_1} \odot \mathbf{b_2}$	$(\mathbf{q_1} + \mathbf{q_2})/2$ $\mathbf{b_1} + \mathbf{b_2}$	-	$\texttt{dot}(\mathbf{q}, \mathbf{e})$
CQD	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$	$\min_{\mathbf{q_2}} d(\mathbf{q_1}, \mathbf{r}, \mathbf{q_2})$ or $\text{top}_k(d(\mathbf{q_1}, \mathbf{r}, \mathbf{e_k}))$	Product: $\mathbf{q_1} \cdot \mathbf{q_2}$ Gödel: $\min(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$ $\max(\mathbf{q_1}, \mathbf{q_2})$	-	$\texttt{dist}(\mathbf{q}, \mathbf{e})$
CQD$^{\mathcal{A}}$	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$ $\mathbf{W} \in \mathbb{R}^{2\times 2d}$	$\theta = \mathbf{W}[\mathbf{q_1}, \mathbf{r}]$ $\text{top}_k[\rho_\theta(d(\mathbf{q_1}, \mathbf{r}, \mathbf{e_k}))]$	Product: $\mathbf{q_1} \cdot \mathbf{q_2}$ Gödel: $\min(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$ $\max(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{1} - \mathbf{q}$ $(1 + \cos(\pi \mathbf{q}))/2$	$\texttt{dist}(\mathbf{q}, \mathbf{e})$
Var2Vec	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$ $\mathbf{W} \in \mathbb{R}^{d\times 2d}$	$\mathbf{q_2} = \mathbf{W}[\mathbf{q_1}, \mathbf{r}]$ $d(\mathbf{q_1}, \mathbf{r}, \mathbf{q_2})$	Product: $\mathbf{q_1} \cdot \mathbf{q_2}$ Gödel: $\min(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$ $\max(\mathbf{q_1}, \mathbf{q_2})$	$1 - \mathbf{q}$	$\texttt{dist}(\mathbf{q}, \mathbf{e})$
QTO	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$ $\mathbf{M} \in [0,1]^{\lvert\mathcal{R}\rvert\times\lvert\mathcal{E}\rvert\times\lvert\mathcal{E}\rvert}$	$d(\mathbf{q_1}, \mathbf{r}, \mathbf{q_2})$ $\text{row}_{q1}(\mathbf{M}_r)$	$\mathbf{q_1} \cdot \mathbf{q_2}$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$	$d(\mathbf{q_1}, \mathbf{r}, \mathbf{q_2})$ $\text{row}_{q1}(\mathbf{1} - \mathbf{M}_r)$	$\texttt{dist}(\mathbf{q}, \mathbf{e})$
LogicE	$\mathbf{q}, \mathbf{e} = ([l_i, u_i],$ $l_i, u_i \in [0,1])_{i=1}^d$ $\mathbf{r} \in \mathbb{R}^d$	$\sigma(\max(0, \max(0,$ $[\mathbf{r}, \mathbf{q}]\mathbf{F}_1)\mathbf{F}_2)\mathbf{F}_3)$	$([\top(l_i^{(1)}, \ldots, l_i^{(n)}),$ $\top(u_i^{(1)}, \ldots, u_i^{(n)})])$ $i = 1 \ldots d$	DM	$([1 - l_i,$ $1 - u_i])$ $i = 1 \ldots d$	$\lVert\mathbf{q} - \mathbf{e}\rVert_1$
FuzzQE	$\mathbf{q}, \mathbf{e} \in [0,1]^d$	$\sigma(\texttt{MLP}_r(\mathbf{q}))$	Product: $\mathbf{q_1} \cdot \mathbf{q_2}$ Gödel: $\min(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$ $\max(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{1} - \mathbf{q}$ -	$\texttt{dot}(\mathbf{q}, \mathbf{e})$
TAR	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{R}^d$	$\mathbf{q} + \mathbf{r}$	$\texttt{Attn}(\mathbf{q_1}, \mathbf{q_2})$	$\max(\mathbf{q_1}, \mathbf{q_2})$	$\mathbf{1} - \mathbf{q}$	$\lVert\mathbf{q} - \mathbf{e}\rVert$
GNN-QE	$\mathbf{q}, \mathbf{r} \in \mathbb{R}^{\lvert\mathcal{E}\rvert}$	$\sigma(\texttt{GNN}(\mathbf{q}, \mathcal{G}))$	$\mathbf{q_1} \cdot \mathbf{q_2}$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$	$\mathbf{1} - \mathbf{q}$	$\texttt{BCE}(\mathbf{q})$
FLEX	$\mathbf{q} = (\theta_f, \theta_l)$ $\theta_f \in [-L, L]^d$ $\mathbf{e} = (\theta_f, \mathbf{0})$ $\theta_l \in [0,1]^d$	$g(\texttt{MLP}([\theta_f + \theta_{f,r};$ $\theta_l + \theta_{l,r}]))$ g gates θ_f, θ_e.	$\theta_f = \sum_i \mathbf{a}_i \theta_{q_i,f}$ $\theta_l = \Pi_i \theta_{q_i,l}$	$\theta_f = \sum_i \mathbf{a}_i \theta_{q_i,f}$ $\theta_l = \sum_i \theta_{q_i,l} -$ $\sum_{1\le i<j\le n} \theta_{q_i,l}\theta_{q_j,l} +$ $\cdots + (-1)^{n-1}\Pi_i \theta_{q_i,l}$	$\theta_f = L \cdot \tanh($ $\texttt{MLP}([\theta_f; \theta_l]))$ $\theta_l = 1 - \theta_l$	$\lVert\theta_f^e - \theta_f^q\rVert_1 +$ θ_l^q
TFLEX	$\mathbf{q}, \mathbf{r} = (\mathbf{q}_f^e, \mathbf{q}_l^e, \mathbf{q}_f^t, \mathbf{q}_l^t)$ $\mathbf{q}_f^e, \mathbf{q}_f^t, \mathbf{e}_f^e \in \mathbb{R}^d$ $\mathbf{e} = (\mathbf{e}_f^e, \mathbf{0}, \mathbf{0}, \mathbf{0})$ $\mathbf{q}_l^e, \mathbf{q}_l^t \in [0,1]^d$	Entity: $g(\texttt{MLP}($ $\mathbf{q} + \mathbf{r} + \mathbf{t}))$ Time: $g(\texttt{MLP}($ $\mathbf{q_1} + \mathbf{r} + \mathbf{q_2}))$	$\sum_i \alpha \mathbf{q}_{i,f}^e, \bigotimes_i(\{\mathbf{q}_{i,l}^e\}),$ $\sum_i \beta \mathbf{q}_{i,f}^t, \bigotimes_i(\{\mathbf{q}_{i,l}^t\})$ $\sum_i \alpha \mathbf{q}_{i,f}^e, \bigotimes_i(\{\mathbf{q}_{i,l}^e\}),$ $\sum_i \beta \mathbf{q}_{i,f}^t, \bigotimes_i(\{\mathbf{q}_{i,l}^t\})$	$\sum_i \alpha \mathbf{q}_{i,f}^e, \bigoplus_i(\{\mathbf{q}_{i,l}^e\}),$ $\sum_i \beta \mathbf{q}_{i,f}^t, \bigotimes_i(\{\mathbf{q}_{i,l}^t\})$ $\sum_i \alpha \mathbf{q}_{i,f}^e, \bigoplus_i(\{\mathbf{q}_{i,l}^e\}),$ $\sum_i \beta \mathbf{q}_{i,f}^t, \bigotimes_i(\{\mathbf{q}_{i,l}^t\})$	$f_{\text{not}}^e(\mathbf{q}_f^e),$ $\ominus(\mathbf{q}_l^e), \mathbf{q}_f^t, \mathbf{q}_l^t$ $\mathbf{q}_f^e, \mathbf{q}_l^e,$ $f_{\text{not}}^t(\mathbf{q}_f^t), \ominus(\mathbf{q}_l^t)$	$\lVert\mathbf{e}_f^e - \mathbf{q}_f^e\rVert_1 +$ $\mathbf{q}_f^e\rVert_1 + \mathbf{q}_l^e$ $\lVert\mathbf{t}_f^t - \mathbf{q}_f^t\rVert_1 +$ $\mathbf{q}_l^t$
ENeSy	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in \mathbb{C}^d$ $\mathbf{p}_q, \mathbf{p}_e \in \{0,1\}^{\lvert\mathcal{E}\rvert}$ $\mathbf{M}_r \in \{0,1\}^{\lvert\mathcal{E}\rvert\times\lvert\mathcal{E}\rvert}$	Neural: $\mathbf{q} \circ \mathbf{r}$ Symb: $g(\mathbf{p}_q \mathbf{M}_r)^\top$ $g = \mathbf{x}/\text{sum}(\mathbf{x})$	$g(\mathbf{p_1} \cdot \mathbf{p_2})$	$g(\mathbf{p_1} + \mathbf{p_2} - \mathbf{p_1} \cdot \mathbf{p_2})$	$g(\frac{\alpha}{\lvert\mathcal{E}\rvert} - \mathbf{p})$	$\lVert\mathbf{q} - \mathbf{e}\rVert$
NQE	$\mathbf{q}, \mathbf{e}, \mathbf{r} \in [0,1]^d$	$\sigma(\texttt{Trf}(\mathbf{q}, \mathcal{G}_q))$	$\mathbf{q_1} \cdot \mathbf{q_2}$	$\mathbf{q_1} + \mathbf{q_2} - \mathbf{q_1} \cdot \mathbf{q_2}$	$\mathbf{1} - \mathbf{q}$	$\texttt{dot}(\mathbf{q}, \mathbf{e})$

differentiable interaction mechanism between the fuzzy space $[0,1]^d$ and unbounded embedding space $\mathbb{R}^d$ (or $\mathbb{C}^d$) where relation representations are often initialized from. That is, re-scaling and squashing of vector values when processing a computation graph might lead to noisy gradients and unstable training which is observed, for instance, by GNN-QE that has to turn off gradients from all but last projection step.

5.3 Decoder

The goal of decoding is to obtain the final set of answers or a ranking of all the entities. It is the final step of the query answering task after processing. Here we categorize the methods into two buckets: *non-parametric* and *parametric*. Parametric methods require a parameterized method to score an entity (or predict a regression target from the processed latents) while non-parametric methods can directly measure the similarity (or distance) between a pair of query and entity on the graph. Most of the methods belong to the non-parametric category as shown in the *Distance* column of processor tables Table 7, Table 8, Table 9, Table 10. For instance, geometric models (Ren et al., 2020; Andresel et al., 2021; Adlakha et al., 2021; Choudhary et al., 2021b; Zhang et al., 2021b) pre-define a distance function between the representation of the query and that of an entity. Commonly employed distance functions are L1 (Hamilton et al., 2018; Ren et al., 2022; Gebhart et al., 2021; Amayuelas et al., 2022; Tang et al., 2022; Xu et al., 2022; Long et al., 2022), L2 (Huang et al., 2022b; Chen et al., 2022b), or their variations (Luus et al., 2021; Lin et al., 2022b;a), cosine similarity (Wang et al., 2019; Mai et al., 2019; Daza & Cochez, 2020; Wang et al., 2023c), dot product (Sun et al., 2020; Ren et al., 2022; Alivanistos et al., 2022; Liu et al., 2022; Bai et al., 2022a; Luo et al., 2023), or naturally model the likelihood of all the entities without the need of a distance function (Arakelyan et al., 2021; Kotnis et al., 2021; Zhu et al., 2022; Pflueger et al., 2022; Bai et al., 2022b; Arakelyan et al., 2023; Wang et al., 2023a). Probabilistic models often employ KL divergence (Ren & Leskovec, 2020; Yang et al., 2022a) or Mahalanobis distance (Choudhary et al., 2021a).

One important direction (orthogonal to the distance function) that current methods largely ignore is how to perform efficient answer entity retrieval over extremely large graphs with billions of entities. A scalable

and approximate nearest neighbor (ANN) search algorithm is necessary. Existing frameworks including FAISS (Johnson et al., 2019) or ScaNN (Guo et al., 2020) provide scalable implementations of ANN. However, ANN is limited to L1, L2 and cosine distance and mostly optimized for CPUs. It is still an open research problem how to design efficient scalable ANN search algorithms for more complex distance functions such as KL divergences so that we can retrieve with much better efficiency for different CLQA methods with different distance functions (preferably, using GPUs).

We conjecture that parametric decoders are to gain more traction in numerical tasks on top of plain entity retrieval for query answering. Such tasks might involve numerical and categorical features on node-, edge-, and graph levels, *e.g.*, training a regressor to predict numerical values for node attributes like *age*, *length*, etc. Besides a parametric decoder gives new opportunities to generalize to inductive settings where we may have unseen entities during evaluation. SE-KGE (Mai et al., 2020) takes a step in this direction by predicting geospatial coordinates of query targets.

5.4 Computation Complexity

Here we analyze the time complexity of different query reasoning models categorized by different operations, including relation projection, intersection, union, negation and answer retrieval after obtaining the representation of the query. We list the asymptotic complexity in Table 11 for methods that perform stepwise encoding of the query graph, and Table 12 for methods (mostly GNN-based) that encode the whole query graph simultaneously including projection and other logic operations.

Table 11: Time complexity of each operation on $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$. Across all methods, we denote the embedding dimension and hidden dimension of MLPs as d, number of layers of MLPs/GNNs as l, number of branches in an intersection operation as i, number of branches in a union operation as u.

Model	Projection	Intersection	Negation	Union	Answer Retrieval	Definitions
GQE GQE+hashing RotatE-m Distmult-m ComplEx-m	$\mathcal{O}(d)$	$\mathcal{O}(ild^2)$	-	-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
CGA	$\mathcal{O}(d^2)$	$\mathcal{O}(id + d^2)$	-	-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
Query2Particles	$\mathcal{O}(Kd^2)$	$\mathcal{O}(iKd^2)$	$\mathcal{O}(Kd^2)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert dK)$	K : #particles.
SignalE	$\mathcal{O}(d)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(d)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
MLPMix	$\mathcal{O}(ld^2)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(ld^2)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
Query2Box Query2Onto	$\mathcal{O}(d)$	$\mathcal{O}(ild^2)$	-	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
Query2Geom	$\mathcal{O}(d)$	$\mathcal{O}(d)$	-	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
RotatE-Box	$\mathcal{O}(d)$	-	-	DNF / $\mathcal{O}(uld^2)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
NewLook	$\mathcal{O}(\lvert\mathcal{E}\rvert d + ld^2)$	$\mathcal{O}(i(\lvert\mathcal{E}\rvert + ld^2))$	$\mathcal{O}(ld^2)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
HypE	$\mathcal{O}(d)$	$\mathcal{O}(ild^2)$	-	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
ConE/BetaE	$\mathcal{O}(ld^2)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(d)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
PERM	$\mathcal{O}(d^2)$	$\mathcal{O}(id^3)$	-	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d^2)$	-
LinE	$\mathcal{O}(ld^2)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
GammaE	$\mathcal{O}(ld^2)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(d)$	$\mathcal{O}(uld^2)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
NMP-QEM	$\mathcal{O}(Kld^2)$	$\mathcal{O}(iK^2d)$	$\mathcal{O}(Kld^2)$	DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert dK)$	K : # centers
EmQL	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	$\mathcal{O}(d)$	-	$\mathcal{O}(d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
CQD-CO	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	Opt	-	Opt	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
CQD-Beam	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert kd)$	-	$\mathcal{O}(\lvert\mathcal{E}\rvert kd)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
QTO	$\mathcal{O}(\max_k \lvert T^*(v_k) > 0\rvert\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$T^*(v)$: the maximum truth value for the subquery rooted at node v.
LogicE	$\mathcal{O}(ld^2)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
FuzzQE	$\mathcal{O}(d^2)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
TAR	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
GNN-QE	$\mathcal{O}(\lvert\mathcal{E}\rvert d^2 + \lvert\mathcal{S}\rvert d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	-
FLEX	$\mathcal{O}(ld^2)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(ld^2)$	$\mathcal{O}(uld^2 + 2^u d)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-
TFLEX	$\mathcal{O}(ld^2)$	$\mathcal{O}(ild^2)$	$\mathcal{O}(ld^2)$	$\mathcal{O}(uld^2)$	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$	-

Table 12: Time complexity of answering a query for methods that directly encode the query graph. Besides the operators defined in Table 11, we denote n_q as average degree of the query graph $\mathcal{G}_q = (\mathcal{E}_q, \mathcal{R}_q, \mathcal{S}_q)$.

Model	Projection	Intersection	Negation	Union	Answer Retrieval
MPQE / GNNQ	$\mathcal{O}(d^2 n_q l \lvert\mathcal{E}_q\rvert)$		-	-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$
StarQE	$\mathcal{O}(d^2 n_q l \lvert\mathcal{E}_q\rvert + \lvert\mathcal{S}_q\rvert \lvert qp\rvert d)$		-	-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$
LMPNN	$\mathcal{O}(d^2 n_q l \lvert\mathcal{E}_q\rvert)$			DNF	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$
BiQE	$\mathcal{O}((\lvert\mathcal{E}_q\rvert d^2 + \lvert\mathcal{E}_q\rvert^2 d)l)$		-	-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$
kgTransformer	$\mathcal{O}((d^2 n_q \lvert\mathcal{E}_q\rvert + \lvert\mathcal{E}_q\rvert n_q^2 d)l)$			-	$\mathcal{O}(\lvert\mathcal{E}\rvert d)$
SQE	$\mathcal{O}((\lvert\mathcal{E}_q\rvert d^2 + \lvert\mathcal{E}_q\rvert^2 d)l)$				$\mathcal{O}(\lvert\mathcal{E}\rvert d)$

6 Queries

The third direction to segment the methods is from the queries point of view. Under the queries category, we have three subcategories: *Query Operators*, *Query Patterns*, and *Projected Variables*. For query operators, methods have different operator expressiveness, which means the set of query operators one model is able to handle including existential quantification ($\exists$), conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), Kleene plus ($+$), filter and various aggregation operators. For query patterns, we refer to the structure/pattern of the (optimized) query plan, ranging from paths and trees to arbitrary directed acyclic graphs (DAGs) and cyclic patterns. As to projected variables (by *projected* we refer to target variables that have to be bound to particular graph elements like entity or relation), queries might have a different number (zero or more) of target variables. We are interested in the complexity of such projections as binding of two and more variables involves relational algebra (Codd, 1970) and might result in a Cartesian product of all retrieved answers.

6.1 Query Operators

Different query processors have different expressiveness in terms of operators a method can handle. Throughout all the works, we compiled Table 13 that classifies all methods based on the supported operators.

Table 13: Query answering processors and supported query operators. Processors supporting unions $\vee$ also support projection and intersection ($\wedge$). Models supporting negation ($\neg$) also support unions, projections, and intersections.

Projection and Intersection ($\wedge$)	Union ($\vee$)	Negation ($\neg$)	Kleene +	Filter & Aggr
GQE (Hamilton et al., 2018)	Query2Box (Ren et al., 2020)	BetaE (Ren & Leskovec, 2020)	RotatE-Box	None
GQE hashed (Wang et al., 2019)	EmQL (Sun et al., 2020)	ConE (Zhang et al., 2021b)	(Adlakha et al., 2021)	
CGA (Mai et al., 2019)	Query2Onto (Andresel et al., 2021)	LogicE (Luus et al., 2021)		
TractOR (Friedman & Broeck, 2020)	HypE (Choudhary et al., 2021b)	MLPMix (Amayuelas et al., 2022)		
MPQE (Daza & Cochez, 2020)	NewLook (Liu et al., 2021)	Query2Particles (Bai et al., 2022a)		
BiQE (Kotnis et al., 2021)	PERM (Choudhary et al., 2021a)	LinE (Huang et al., 2022b)		
Sheaves (Gebhart et al., 2021)	CQD (Arakelyan et al., 2021)	GammaE (Yang et al., 2022a)		
StarQE (Alivanistos et al., 2022)	kgTransformer (Liu et al., 2022)	NMP-QEM (Long et al., 2022)		
RotatE-m, DistMult-m,	Query2Geom (Sardina et al., 2023)	FuzzQE (Chen et al., 2022b)		
ComplEx-m (Ren et al., 2022)		TAR (Tang et al., 2022)		
GNNQ (Pflueger et al., 2022)		GNN-QE (Zhu et al., 2022)		
		FLEX (Lin et al., 2022b)		
		TFLEX (Lin et al., 2022a)		
		ENeSy (Xu et al., 2022)		
		QTO (Bai et al., 2022b)		
		SignalE (Wang et al., 2022a)		
		LMPNN (Wang et al., 2023c)		
		NQE (Luo et al., 2023)		
		Var2Vec (Wang et al., 2023a)		
		CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023)		
		SQE (Bai et al., 2023)		

We start from simplest conjunctive queries that involve only existential quantification ($\exists$) and conjunction ($\wedge$) and gradually increase the complexity of supported operators.

Existential Quantification (∃). When ∃ appears in a query, this means that there exists at least one existentially quantified variable. For example, given a query "At what universities do the Turing Award winners work?" and its logical form $q = U_? \,.\, \exists V \,:\, win(\texttt{TuringAward}, V) \wedge university(V, U_?)$, here V is the existentially quantified variable. Query processors model existential quantification by using a relation projection operator. Mapping to query languages like SPARQL, relation projection is equivalent to a *triple pattern* with one variable, *e.g.*, `{TuringAward win ?v}` (Fig. 18). Generally, as introduced in Section 5.2, query embedding methods embed a query in a bottom-up fashion, starting with the embedding of the anchors (leaf nodes) and gradually traversing the query tree up to the root. In such a way, query embedding methods (*e.g.*, geometric or probabilistic) explicitly obtain an embedding for the existentially quantified variables. The embeddings/representations of these variables are calculated by a relation projection function implemented as shallow vector operations (Hamilton et al., 2018; Ren et al., 2020; Choudhary et al., 2021b; Arakelyan et al., 2021; Ren et al., 2022) or deep neural nets (Ren & Leskovec, 2020; Zhang et al., 2021b; Bai et al., 2022a; Amayuelas et al., 2022). Another set of methods based on GNNs and Transformers directly assigns a learnable initial embedding for the existentially quantified variables. These embeddings are then updated through several message passing layers over the query plan (Daza & Cochez, 2020; Alivanistos et al., 2022; Pflueger et al., 2022; Wang et al., 2023c) or attention layers over the serialized query plan (Kotnis et al., 2021; Liu et al., 2022).

The major drawback of existing neural query processors is the assumption of at least one anchor entity in the query from which the answering process starts and relation projections can be executed. It remains an open challenge and an avenue for future work to support queries without anchor entities, *e.g.*, $q_1 = U_?.\exists v_1, v_2 \,:\, win(v_1, v_2) \wedge university(v_2, U_?)$, and queries where relations are variables, *e.g.*, $q_2 = r_? : r_?(\texttt{TuringAward}, \texttt{Bengio})$, that can be framed as the relation prediction task.

Universal Quantification (∀). A universally quantified variable $\forall x.P(x)$ means that a logical formula $P(x)$ holds for all possible x. Usually, the universal quantifier does not appear in facts-only ABox graphs without types and complex axioms (see Section 4.3) as it would imply that some entity is connected to all other entities. For example, $\forall V.win(\texttt{TuringAward}, V)$ without other constraints implies that all entities are connected to `TuringAward` by the *win* relation (which does not occur in practice). However, universal quantifiers are more useful when paired with the class hierarchy (unary relations), *e.g.*, $\exists\, ?paper, \forall r \in \texttt{Researcher} : \texttt{Researcher}(r) \wedge authored(r, ?paper)$ means that the *authored* relation projection would be applied only to entities of class `Researcher`.

Currently, existing CLQA approaches do not support universal quantifiers explicitly nor the datasets include queries with the ∀ quantifier. Still, using the basic identity $\forall x.P(x) \equiv \neg(\exists x.\neg P(x))$ it is possible to model the universal quantifier by any approach supporting existential quantification ∃ and negation ¬. By default, we assume the closed-world assumption in NGDBs. We leave the implications of universal quantification pertaining to the open-world assumption (OWA) out of the scope of this work.

Conjunction (∧). We elaborate on the differences of those query patterns in the following Section 6.2 and emphasize our focus on more complex intersection queries going beyond simpler path-like queries.

Query processors, as described in Section 5.2, employ different parameterizations of conjunctions as permutation-invariant set functions. A family of neural processors (Hamilton et al., 2018; Wang et al., 2019; Ren et al., 2022) often resort to the DeepSet architecture (Zaheer et al., 2017) that first projects each set element independently and then pools representations together with a permutation-invariant function (*e.g.*, sum, mean) followed by an MLP. Alternatively, (Mai et al., 2019; Bai et al., 2022a) self-attention, can serve as a replacement of the DeepSet where set elements are weighted with the attention operation. The other family of neural processors combine projection and intersection by processing the whole query graph with GNNs (Daza & Cochez, 2020; Alivanistos et al., 2022; Pflueger et al., 2022; Wang et al., 2023c) or with the Transformer over linearized query sequences (Kotnis et al., 2021; Liu et al., 2022). Geometric processors (Ren et al., 2020; Liu et al., 2021; Choudhary et al., 2021b; Zhang et al., 2021b) implement conjunction as the attention-based average of centroids and offsets of respective geometric objects (boxes, hyperboloids, or cones). Probabilistic processors (Ren & Leskovec, 2020; Choudhary et al., 2021a; Huang

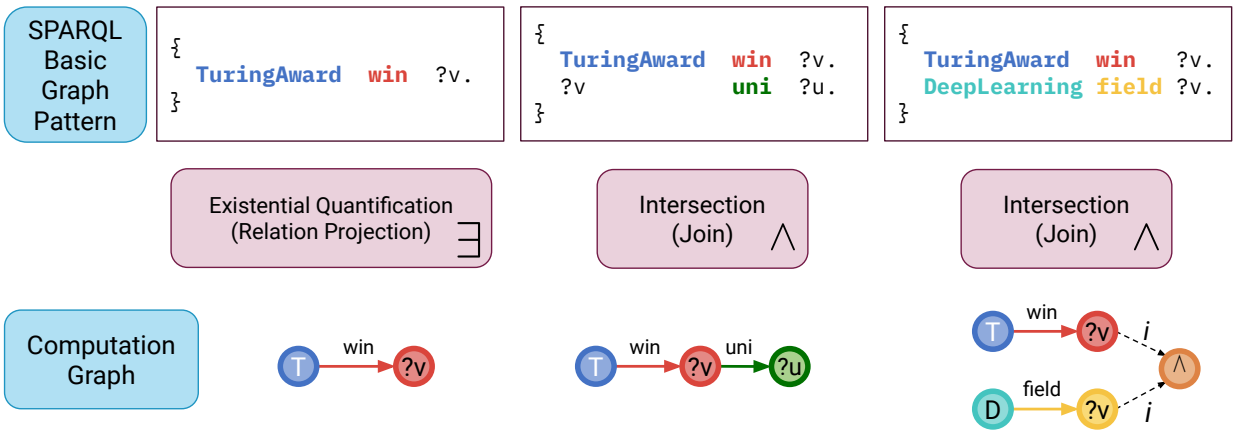

Figure 18: Query operators (relation projection and intersection), corresponding SPARQL basic graph patterns (BGP), and their computation graphs. Relation Projection (left) corresponds to a triple pattern.

et al., 2022b; Yang et al., 2022a) implement intersection as a weighted sum of parametric distributions that represent queries and variables.

Fuzzy-logic processors (Arakelyan et al., 2021; Luus et al., 2021; Chen et al., 2022b; Zhu et al., 2022) commonly resort to *t-norms*, generalized versions of conjunctions in the continuous $[0, 1]$ space and corresponding *t-conorms* for modeling unions (Section 2.6). Often, due to the absence of a principal study, the choice of the fuzzy logic is a hyperparameter. We posit that such a study is an important avenue for future works in fuzzy processors. More exotic neuro-symbolic methods for modeling conjunctions include element-wise product of count-min sketches (Sun et al., 2020) or as a weighted sum in the *feature logic* (Lin et al., 2022b;a). Finally, some processors (Tang et al., 2022; Xu et al., 2022) perform conjunctions both in the embedding and symbolic space with neural and fuzzy operators.

We note that certain neural query processors that embed queries directly (Hamilton et al., 2018; Wang et al., 2019; Mai et al., 2019; Friedman & Broeck, 2020; Ren et al., 2022) or via GNN/Transformer encoder over a query graph (Daza & Cochez, 2020; Kotnis et al., 2021; Gebhart et al., 2021; Alivanistos et al., 2022; Das et al., 2022; Pflueger et al., 2022) support only projections and intersections, that is, their extensions to more complex logical operators are non-trivial and might require changing the underlying assumptions of modeling entities, variables, and queries. In some cases, support of unions might be enabled when re-writing a query to the disjunctive normal form (discussed below).

Disjunction ($\vee$). Query processors implement the disjunction operator in several ways. However, modeling disjunction is notoriously hard since it requires modeling any powerset of entities[7] on the graph in a vector space. Before delving into details about different ways of modeling disjunction, we first refer the readers to the Theorem 1 in Query2Box (Ren et al., 2020). The theorem proves that we need the VC dimension of the function class of the distance function to be around the number of entities on the graph.

The theorem shows that in order to accurately model *any* EPFO query with the existing framework, the complexity of the distance function measured by the VC dimension needs to be as large as the number of KG entities. This implies that if we use common distance functions based on hyper-plane, Euclidean sphere, or axis-aligned rectangle,[8] their parameter dimensionality needs to be $\Theta(M)$, which is $\Theta(|\mathcal{E}|)$ for real KGs we are interested in. In other words, the dimensionality of the logical query embeddings needs to be $\Theta(|\mathcal{E}|)$, which is not low-dimensional; thus not scalable to large KGs and not generalizable in the presence of unobserved KG edges.

[7]Here the most rigorous description should be powerset of sets of "isomorphic" entities on the graph. By "isomorphic" entities, it refer to an entity set where each elemen

[8]For the detailed VC dimensions of these function classes, see Vapnik (2013). Crucially, their VC dimensions are all linear with respect to the number of parameters d.

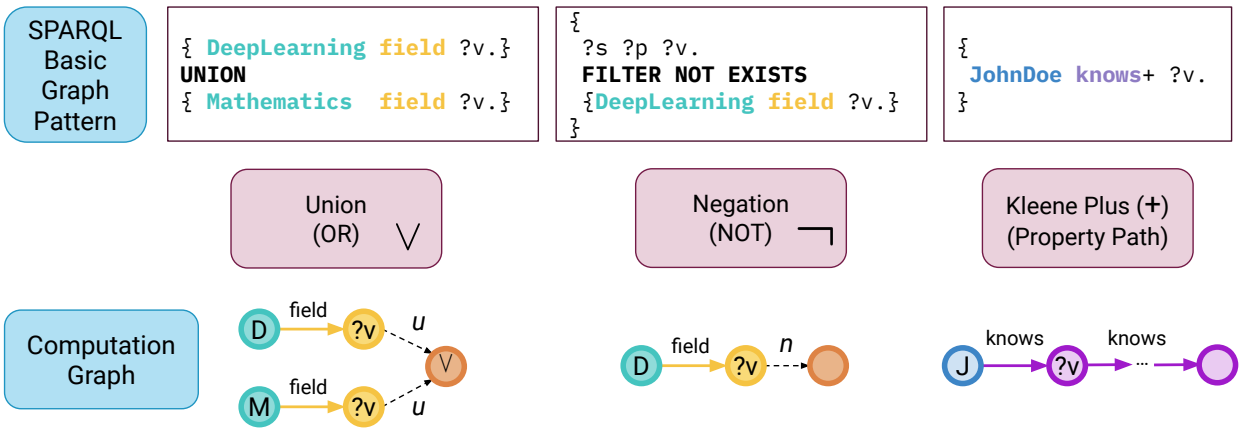

Figure 19: Query operators (union, negation, Kleene plus), corresponding SPARQL basic graph patterns (BGP), and their computation graphs.

The first idea proposed in Query2Box (Ren et al., 2020) is that given a model has defined a distance function between a query representation and the entity representation, then a query can be transformed (or re-written) into its equivalent *disjunctive normal form* (DNF), *i.e.*, a disjunction of conjunctive queries. For example, we can safely convert a query $(A \vee B) \wedge (C \vee D)$ to $((A \wedge C) \vee (A \wedge D) \vee (B \wedge C) \vee (B \wedge D))$, where A, B, C, D are atomic formulas. In such a way, we only need to process disjunction $\vee$ at the very last step. For models that have defined a distance function between the query representation and entity representation $d(\mathbf{q}, \mathbf{e})$ (such as geometric processors (Ren et al., 2020; Andresel et al., 2021; Adlakha et al., 2021; Choudhary et al., 2021b; Zhang et al., 2021b; Sardina et al., 2023) and some neural processors (Liu et al., 2022; Amayuelas et al., 2022; Wang et al., 2023c), the idea of using DNF to handle disjunction is to (1) embed each atomic formula / conjunctive query in the DNF into a vector $\mathbf{q_i}$, (2) calculate the distance between the representation/embedding of each atomic formula / conjunctive query and the entity $d(\mathbf{q_i}, \mathbf{e})$, (3) take the minimum of the distances $min_i(d(\mathbf{q_i}, \mathbf{e}))$. The intuition is that since disjunction models the union operation, as long as the node is close to one atomic formula / conjunctive query, it should be close to the whole query. Potentially, many neural processors with the defined distance function and originally supporting only intersection and projection can be extended to supporting unions with DNF. One notable downside of this modeling is that it is exponentially expensive (to the number of disjunctions) in the worst case when converting a query to its DNF.

Another category of mostly probabilistic processors (Choudhary et al., 2021a; Yang et al., 2022a) proposes a neural disjunction operator implemented with the permutation-invariant attention over the input set with the *closure* assumption that the result of attention weighting union remains in the same probabilistic space as its inputs. Such models design a more black-box framework to handle the disjunction operation under the strong closture assumption that might not be true in all cases.

The third way of modeling disjunction is based on the De Morgan's laws (Ren & Leskovec, 2020). According to the De Morgan's laws (DM), the disjunction is equivalent to the negation of the conjunction of the negation of the statements making up the disjunction, *i.e.*, $A \vee B = \neg(\neg A \wedge \neg B)$. For methods that can handle the negation operator (detailed in the following paragraph), they model disjunction by using three negation operations and one conjunction operation. DM conversion was explicitly probed in probabilistic (Ren & Leskovec, 2020), geometric (Zhang et al., 2021b), and fuzzy (Luus et al., 2021) processors.

Finally, most fuzzy-logic (Arakelyan et al., 2021; Chen et al., 2022b; Tang et al., 2022; Zhu et al., 2022; Bai et al., 2022b; Arakelyan et al., 2023; Wang et al., 2023a) processors employ *t-conorms*, generalized versions of disjunctions in the continuous $[0, 1]$ space (Section 2.6). More exotic versions of neuro-symbolic disjunctions include element-wise summation of count-min sketches (Sun et al., 2020), feature logic operations (Lin et al., 2022b;a), as well as performing a union in both embedding and symbolic spaces (Tang et al., 2022; Xu et al., 2022) with fuzzy operators.

Negation ($\neg$). For negation operation, the goal is to model the complement set, *i.e.*, the answers $\mathcal{A}_q$ to a query $q = V_? : \neg r(v, V_?)$ are the exact complement of the answers $\mathcal{A}_{q'}$ to query $q' = V_? : r(v, V_?)$: $\mathcal{A}_q = \mathcal{V}/\mathcal{A}_{q'}$. Correspondingly, negation in SPARQL can be implemented with `FILTER NOT EXISTS` or `MINUS` clauses. For example (Fig. 19), a logical formula with negation $\neg$*field*(`DeepLearning, V`) is equivalent to the SPARQL BGP `{?s ?p ?v. FILTER NOT EXISTS {DeepLearning field ?v}}` where `{?s ?p ?v}` models the *universe set* ($\mathbf{1}$) of all facts that gets filtered by the triple pattern.

Modeling the *universe set* ($\mathbf{1}$) and its complement is the key problem when designing a negation operator in neural query processors, *e.g.*, an arbitrary real $\mathbb{R}$ or complex $\mathbb{C}$ space is unbounded such that $\mathbf{1}$ is not defined. For that reason, many neural processors do not support the negation operator. Still, there exist several approaches to handle negation.

The first line of works (Bai et al., 2022a; Amayuelas et al., 2022; Long et al., 2022) designs a purely neural MLP-based negation operator over the query representation avoiding the universe set altogether. Similarly, a token of the negation operator can be included into the linearized query representation (Bai et al., 2023) to be encoded with Transformer or recurrent network. A step aside from purely neural operators is taken by GNN-based processors (Wang et al., 2023c) that treat a negation edge as a new edge type during message passing over the query computation graph.

The second line is customized to different embedding spaces and aims to simulate the calculation of the universe and complement in the embedding space, *e.g.*, using geometric cones (Zhang et al., 2021b), parameters are angles θ such that the space (and, hence, $\mathbf{1}$) is bounded to 2π and the complement is straight $2\pi - \theta$. Probabilistic methods (Ren & Leskovec, 2020; Huang et al., 2022b; Yang et al., 2022a) naturally represent negation as an inverse of distribution parameters.

Thirdly, fuzzy logic processors explicitly model the universe set $\mathbf{1}$ and the complement over the same real valued logic space. For instance, LogicE (Luus et al., 2021) and FuzzQE (Chen et al., 2022b) restrict the query embedding space to the range $[0, 1]^d$ where each query $\mathbf{q} \in [0, 1]^d$ is a vector. This way, the universe $\mathbf{1}$ is represented with a vector of all ones (in the embedding space $\mathbf{1}^d$) and negation is simply $\mathbf{1} - \mathbf{q}$. TAR (Tang et al., 2022) and GNN-QE (Zhu et al., 2022) operate over fuzzy sets where each entity has a corresponding scalar $\mathbf{q} \in [0, 1]$ in the bounded range. Therefore, the universe $\mathbf{1}$ can still be a vector of all ones (in the entity space $\mathbf{1}^{|\mathcal{E}|}$) and negation is $\mathbf{1} - \mathbf{q}$. ENeSy (Xu et al., 2022) defines the universe as the uniform distribution over the entity space with each element weighting $\frac{\alpha}{|\mathcal{E}|}$ (α is a hyperparameter). CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023) employs a strict cosine fuzzy negation $\frac{1}{2}(1 + \cos(\pi\mathbf{q}))$ over scalar scores $\mathbf{q}$. More exotic processors (Lin et al., 2022b;a) employ feature logic for modeling negation. We also note that the *difference* operator introduced in Liu et al. (2021) is in fact a common intersection-negation (*2in*) query pattern used in all standard benchmarks (Section 7).

Kleene Plus (+) and Property Paths. Kleene Plus is an operator that applies compositionally and recursively to any regular expression (RegEx) that denotes *one or more* occurrence of the specified pattern. Regular expressions exhibit a direct connection to *property paths* in SPARQL. We defined a very basic regular graph query in Definition 2.12, here we generalize that further to property paths. To define property paths more formally, given a set of relations $\mathcal{R}$ and operators $\{+, *, ?, !, \hat{}, /, |\}$, a property path p can be obtained from the recursive grammar $p ::= r \mid p^+ \mid p^* \mid p? \mid !p \mid \hat{p} \mid p_1/p_2 \mid$ "$p_1|p_2$" Here, r is any element of $\mathcal{R}$, $+$ is a Kleene Plus denoting *one or more* occurrences, $*$ is a Kleene Star denoting *zero or more* occurrences, ? denotes *zero or one* occurences, ! denotes negation of the relation or path, $\hat{p}$ traverses an edge of type p in the opposite direction, p_1/p_2 is a sequence of relations (corresponds to *relation projection*), and $p_1|p_2$ denotes an alternative path of p_1 or p_2 (corresponds to a *union* operation). For example (Fig. 19), an expression with Kleene plus *knows*(`JohnDoe`, $V)^+$ can be represented as a SPARQL property path `{JohnDoe knows+ ?v.}`.

Property paths are non-trivial to model for neural query processors due to compositionalilty and recursive nature. To the best of our knowledge, RotatE-Box (Adlakha et al., 2021) is the only geometric processor that handles the Kleene Plus operator implementing the subset of operators $\{+, /, |\}$. RotatE-Box provides two ways to handle Kleene Plus. The first method is to define a $\mathbf{r}^+$ embedding for each relation $r \in \mathcal{R}$, note this is independent and separate from the regular relation embedding for r; another way is to use a trainable matrix to transform the relation embedding $\mathbf{r}$ to the $\mathbf{r}^+$ embedding. Note the two methods do

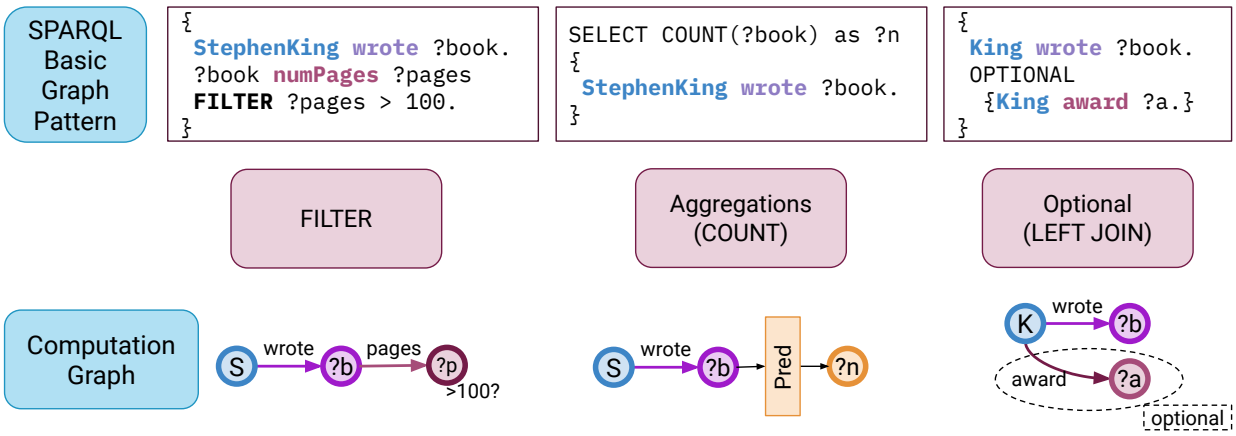

Figure 20: Query operators (`Filter`, `Count` Aggregation, `Optional`), corresponding SPARQL basic graph patterns (BGP), and their computation graphs.

not support Kleene Plus over paths. RotatE-Box also implements relation projection (as a rotation in the complex space) and union (with DeepSets or DNF) but does not support the intersection operator.

We hypothesize that better support of the property paths vocabulary might be one of main focuses in future neural query processors. Particularly for Kleene Plus, some unresolved issues include supporting idempotence ($(r^+)^+ = r^+$) and infinite union of sets ($r^+ = r|(r/r)|(r/r/r)\dots$).

Filter. Filter is an operation that can be inserted in a SPARQL query. It takes any expression of boolean type as input and aims to filter the results based on the boolean value, *i.e.*, only the results rendered `True` under the expression will be returned. The boolean expression can thus be seen as a condition that the answers to the query should follow. For the filter operator, we can do filter on values/literals/attributes, *e.g.*, $\texttt{Filter}(V_{\text{date}} \geq \text{“}2000-01-01\text{”} \&\& V_{\text{date}} \leq \text{“}2000-12-31\text{”})$ means we would like to filter dates not in the year 2000; $\texttt{Filter}(\texttt{LANG}(V_{\text{book}}) = \text{“}en\text{”})$ means we would like to filter books not written in English, $\texttt{Filter}(?\text{pages} > 100)$ means returning the books that have more than 100 pages (as illustrated in Fig. 20). To the best of our knowledge, there does not exist a reasoning model that claims to handle Filters, which leaves room for future work on this direction.

We envision several possibilities to support filtering in neural query engines: (1) the simplest option used by Thorne et al. (2021a;b) in natural language engines is to defer filtering to the postprocessing stage when the set of candidate nodes is identified and their attributes can be extracted by a lookup. (2) Filtering often implies reasoning over literal values and numerical node attributes, that is, processors supporting continuous values (as described in Section 4.1) might be able to perform filtering in the latent space by attaching, for instance, a parametric regressor decoder (Section 5.3) when predicting $?\text{pages} > 100$.

Aggregation. Aggregation is a set of operators in SPARQL queries including `COUNT` (return the number of elements), `MIN`, `MAX`, `SUM`, `AVG` (return the minimum / maximum / sum / average value of all elements), `SAMPLE` (return any sample from the set). For example (Fig. 20), given a triple pattern `{StephenKing wrote ?book.}`, the clause `COUNT (?book) as ?n)` returns the total number of books written by `StephenKing`.

Most aggregation operators require reasoning over sets of numerical values/literals. Such symbolic operations have long been considered a challenge for neural models (Hendrycks et al., 2021). How to design a better representation for numerical values / literals requires remains an open question. Some neural query processors (Ren et al., 2020; Ren & Leskovec, 2020; Zhang et al., 2021b; Zhu et al., 2022), however, have the means to estimate the cardinality of the answer set (including predicted hard answers) that directly corresponds to the `COUNT` aggregation over the target projected variable (assumed to be an entity, not a literal). For example, GNN-QE (Zhu et al., 2022) returns a fuzzy set, *i.e.*, a scalar likelihood value for each entity, that, after thresholding, has low mean absolute percentage error (MAPE) of the number of ground

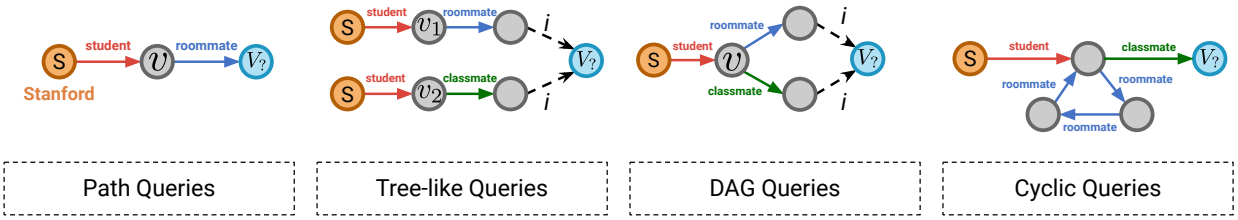

Figure 21: Query patterns: path, tree-like, DAG, and cyclic queries. A DAG query has two branches from the intermediate variable, a cyclic query contains a 3-cycle. Existing neural query processors support path and tree-like patterns.

truth answers. Answer cardinality estimation is thus obtained as a byproduct of the neural query processor without tailored predictors. Alternatively, when models cannot predict the exact count, Spearman's rank correlation is a surrogate metric to evaluate the correlation between model predictions and the exact count. Spearman's rank correlation and MAPE of the number of ground truth answers are common metrics to evaluate the performance of neural query processors and we elaborate on the metrics in Section 7.5.

Optional and Solution Modifiers. SPARQL offers many features yet to be incorporated into neural query engines to extend their expressiveness. Some of those common features include the `OPTIONAL` clause that is essentially a `LEFT JOIN` operator. For example (Fig. 20), given a triple pattern `{King wrote ?book.}` that returns books, the optional clause `{King wrote ?book. OPTIONAL {King award ?a.}}` enriches the answer set with any existing awards received by `King`. Importantly, if there are no bindings to the optional clause, the query still returns the values of `?book`. In the query's computation graph, the optional clause corresponds to the optional branch of a relation projection. A particular challenge for neural query processors operating on incomplete graphs is that the absence of the queried edge in the graph does not mean that there are no bindings – instead, the edge might be missing and might be predicted during query processing.

Solution modifiers, *e.g.*, `GROUP BY`, `ORDER BY`, `LIMIT`, apply further postprocessing of projected (returned) results and are particularly important when projecting several variables in the query. So far, all existing neural query processors are tailored for only one return variable. We elaborate on this matter in Section 6.3.

A General Note on Incompleteness. Finally, we would like to stress out that neural query engines performing all the described operators (**Projection**, **Intersection**, **Union**, **Negation**, **Property Paths**, **Filters**, **Aggregations**, **Optionals**, and **Modifiers**) assume the underlying graph is incomplete and queries might have some missing answers to be predicted, hence, all the operators should incorporate predicted *hard answers* in addition to *easy answers* reachable by graph traversal as in symbolic graph databases. Evaluation of query performance with those operators in light of incompleteness is still an open challenge (we elaborate on that in Section 7.5), *e.g.*, having an *Optional* clause, it might be unclear when there is no true answer (even predicted ones are in fact false) or a model is not able to predict them.

6.2 Query Patterns

Here, we introduce several types of query patterns commonly used in practical tasks and sort them in the increasing order of complexity. Starting with chain-like *Path* queries known in the literature for years, we move to *Tree-Structured* queries (the main supported pattern in modern CLQA systems). Then, we overview *DAG* and *cyclic* patterns which currently are not supported by any neural query answering system and represent a solid avenue for future work.

Path Queries. As introduced in Section 2.4, previous literature starts with path queries (*aka* multi-hop queries), where the goal is simply to go beyond one-hop queries such as $q = V_?.r(v, V_?)$, where $r \in \mathcal{R}, v \in \mathcal{V}$ and $V_?$ represents the answer variable. As shown in Fig. 4 and Fig. 21, there is no logical operator such as branch intersection or union involved. Therefore, in order to answer such a query, we simply find or infer the

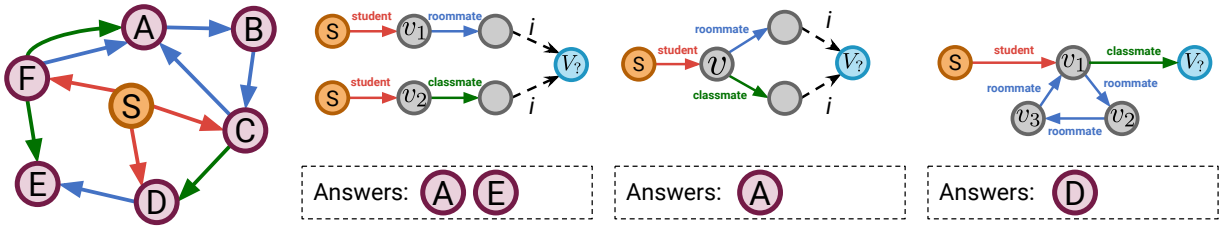

Figure 22: Answers to example tree-like, DAG, and cyclic query patterns given a toy graph. Note the difference in the answer set to the tree-like and DAG queries – in the DAG query, a variable v must have two outgoing edges from the same node.

neighbors of the entity v with relation r. Path queries are a natural extension of one-hop queries. Formally, we denote a path query as follows. $q_{\text{path}} = V_? . \exists V_1, \ldots, V_{k-1} : r_1(v, V_1) \wedge r_2(V_1, V_2) \wedge \cdots \wedge r_k(V_{k-1}, V_?)$, where $r_i \in \mathcal{R}, \forall i \in [1, k]$, $v \in \mathcal{V}$, V_i are all existentially quantified variables. We denote a k-hop path query if it has k atomic formulas. The query plan of a k-hop path query is a chain of length k starting from the anchor entity. For example (Fig. 21), a 2-hop path query is $V_? . \exists v : \textit{student}(\texttt{Stanford}, v) \wedge \textit{roommate}(v, V_?)$ where `Stanford` is the starting anchor node, v is a tail variable of the first projection *student* and at the same time is the head variable of the second projection *roommate* thus forming a chain.

As shown in the definition, in order to handle path queries, it is necessary to develop a method to handle existential quantification $\exists$ and conjunction $\wedge$ operators. Several query reasoning methods (Guu et al., 2015; Das et al., 2017) aim to answer the path queries with sequence models using either chainable KG embeddings (*e.g.*, TransE) in Guu et al. (2015) or LSTM in Das et al. (2017). These methods initiated one of the first efforts that use embeddings and neural methods to answer multi-hop path queries. We acknowledge the efforts in this domain but emphasize their limitations in terms of query expressiveness and, therefore, focus our attention in this work on more expressive query answering methods that operate on tree-like and more complex patterns.

Tree-Structured Queries. Path queries only have one anchor entity and one answer variable. Such queries have limited expressiveness and are far away from real-world query complexity seen in the logs (Malyshev et al., 2018) of real-world KGs like Wikidata. One direct extension to increase the expressiveness and complexity is to support tree-structured (tree-like) queries. Tree-like queries may have multiple anchor entities, and different branches (from different anchors) will merge at the final single answer node, thus forming a tree structured query plan. Such merge can be achieved by intersection, union, or negation operators. For example, as shown in Fig. 1, the query plan of "At what universities do the Turing Award winners in the field of Deep Learning work?" is not a path but a tree. Alternatively, the example in Fig. 21 depicts a query $q = V_?, \exists v_1, v_2 : \textit{student}(\texttt{Stanford}, v_1) \wedge \textit{roommate}(v_1, V_?) \wedge \textit{student}(\texttt{Stanford}, v_2) \wedge \textit{classmate}(v_2, V_?)$ that consists of two branches of 2-hop path queries joined by the intersection operator at the end.

Tree-like queries pose more challenges to the previous models that are only able to handle path (multi-hop) queries since a sequence model no longer applies to tree-structured execution plans with logical operators. In light of the challenges, neural and neuro-symbolic query processors (described in Section 5) are designed to execute more complex query patterns. These processors design neural set/logic operators and do a bottom-up traversal of the tree up to the single root node.

Arbitrary DAGs. Based on tree-structured queries, one can further increase the complexity of the query pattern to arbitrary directed acyclic graphs (DAGs). The key difference between the two types of queries is that for DAG-structured queries, one variable node in the query plan (that represents a set of entities) may be split and routed to different reasoning paths, while the number of branches/reasoning paths in the query plan always decreases from the anchor nodes to the answer node. We show one example in Fig. 21 and in Fig. 22. Consider the tree-like query from the previous paragraph $q_1 = V_?, \exists v_1, v_2 : \textit{student}(\texttt{Stanford}, v_1) \wedge \textit{roommate}(v_1, V_?) \wedge \textit{student}(\texttt{Stanford}, v_2) \wedge \textit{classmate}(v_2, V_?)$ and the DAG query $q_2 = V_?, \exists v : \textit{student}(\texttt{Stanford}, v) \wedge \textit{roommate}(v, V_?) \wedge \textit{classmate}(v, V_?)$. The two queries search for $V_?$ who

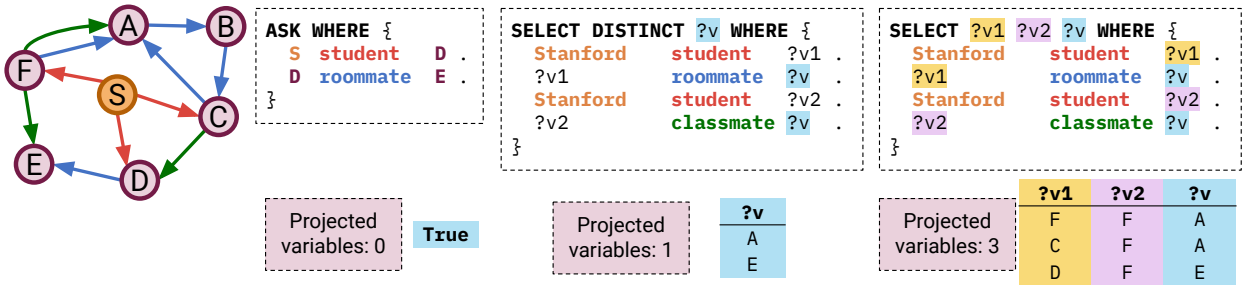

Figure 23: Projected variables of the tree-like query from Fig. 22. Current neural query processors support the single-variable `DISTINCT` mode (center) whereas queries might have zero return variables akin to a subgraph matching Boolean `ASK` query (left) or multiple projected variables (right) that imply returning intermediate answers and form output tuples.

are *roommate* and *classmate* with `Stanford` students. However, the answer sets of the two queries are different (illustrated in Fig. 22). That is, the answer to the DAG query $V_{q_2} = \{\texttt{A}\}$ is the subset of the answers to the tree-like query $V_{q_1} = \{\texttt{A,E}\}$ because the answers to q_2 have to be both *roommate* and *classmate* with the **same** Stanford student in the intermediate variable v. On the other hand, the two branches of the tree-like query q_1 are independent such that intermediate variables v_1 and v_2 need not be the same entities, hence, the query has more valid intermediate answers and more correct answers. To the best of our knowledge, there still does not exist a neural query processor that can faithfully handle any DAG query. Although BiQE (Kotnis et al., 2021) claims to support DAG queries, the mined dataset consists of tree-like queries. Nevertheless, we hypothesize that, potentially, processors with message passing or Transformer architectures that consider the entire query graph structure $\mathcal{G}_q$ may be capable of handling DAG queries and leave this question for future work.

Cyclic Queries. Cyclic queries are more complex than DAG-structured queries. A cycle in a query naturally entails no particular order to traverse the query plan. An example of the cyclic query is illustrated in Fig. 21 and Fig. 22: $q = V_?, \exists v_1, v_2, v_3 : \textit{student}(\texttt{Stanford}, v_1) \wedge \textit{roommate}(v_1, v_2) \wedge \textit{roommate}(v_2, v_3) \wedge \textit{roommate}(v_3, v_1) \wedge \textit{classmate}(v_1, V_?)$. In q, three variables form a triangle cycle $\textit{roommate}(v_1, v_2) \wedge \textit{roommate}(v_2, v_3) \wedge \textit{roommate}(v_3, V_?)$. Given a graph in Fig. 22, the cycle starts and ends at node `C`, hence the only correct answer is obtained after performing the *classmate* relation projection from `C` ending in `D`, $V_q = \{\texttt{D}\}$.

Reasoning methods and query processors that assume a particular traversal or node ordering on the query plan, therefore, cannot faithfully answer cyclic queries. It remains an open question how to effectively model a query with cyclic structures. Moreover, cyclic structures often appear when processing queries with regular expressions and *property paths* (Section 6.1). We posit that supporting cycles might be a necessary condition to fully enable property paths in neural query engines.

6.3 Projected Variables

By *projected variables* we understand target query variables that have to be bound to particular graph elements such as entity, relation, or literals. For example, a query in Fig. 1 $q = V_?.\exists v : \textit{win}(\texttt{TuringAward}, v) \wedge \textit{field}(\texttt{DeepLearning}, v) \wedge \textit{university}(v, V_?)$ has one projected variable $V_?$ that can be bound to three answer nodes in the graph, $V_? = \{\texttt{UofT}, \texttt{UdeM}, \texttt{NYU}\}$. In the SPARQL literature (Hawke et al., 2013), the `SELECT` query specifies which existentially quantified variables to project as final answers. The pairs of projected variables and answers form *bindings* as the result of the `SELECT` query. Generally, queries might have zero, one, or multiple projected variables, and we align our categorization with this notion. Examples of such queries and their possible answers are provided in Fig. 23. Currently, most neural query processors focus on the setting where queries have only one answer variable – the leaf node of the computation graph, as shown in Fig. 23 (center).

Zero Projected Variables. Queries with zero projected variables do not return any bindings but rather probe the graph on the presence of a certain subgraph or relational pattern where the answer is Boolean `True` or `False`. In SPARQL, the equivalent of zero-variable queries is the `ASK` clause. Zero-variable queries might have all entities and relations instantiated with constants, *e.g.*, $q = \mathit{student}(\mathtt{S}, \mathtt{D}) \wedge \mathit{roommate}(\mathtt{D}, \mathtt{E})$ as in Fig. 23 (left) is equivalent to the SPARQL query `ASK WHERE {S student D. D roommate E.}`. The query probes whether a graph contains a particular subgraph (path) induced by the constants. Such a path exists, so the answer is $q = \{\mathtt{True}\}$.

Alternatively, zero-variable queries might have existentially quantified variables that are never projected (up to the cases where all subjects, predicates, or objects are variables). For example, a query $q_1 = \exists v_1, v_2, v_3 : \mathit{student}(v_1, v_2) \wedge \mathit{roommate}(v_2, v_3)$ probes whether there exist any nodes forming a relational path $v_1 \xrightarrow{\mathit{student}} v_2 \xrightarrow{\mathit{roommate}} v_3$. In a general case, a query $q_2 = \exists p, s, o : p(s, o)$ asks if a graph contains at least one edge.

We note that in the main considered setting with incomplete graphs and missing edges zero-variable queries are still non-trivial to answer. Particularly, a subfield of *neural subgraph matching* (Rex et al., 2020; Huang et al., 2022a) implies having incomplete graphs. We hypothesize such approaches might be found useful for neural query processors to support answering zero-variable queries.

One Projected Variable. Queries with one projected variable return bindings for one (of possibly many) existentially quantified variable. In SPARQL, the projected variable is specified in the `SELECT` clause, *e.g.*, `SELECT DISTINCT ?v` in Fig. 23 (center). Although SPARQL allows projecting variables from any part of a query, most neural query engines covered in Section 5 follow the task formulation of GQE (Hamilton et al., 2018) and allow the projected target variable to be only the **leaf node** of the query computation graph. This limitation is illustrated in Fig. 23 (center) where the target variable $?v$ is the leaf node of the query graph and has two bindings $v = \{\mathtt{A}, \mathtt{E}\}$.

It is worth noting that existing neural query processors are designed to return a *unique set* of answers to the input query, *i.e.*, it corresponds to the `SELECT DISTINCT` clause in SPARQL. In contrast, the default `SELECT` returns *multisets* with possible duplicates. For example, the same query in Fig. 23 (center) without `DISTINCT` would have bindings $v = \{\mathtt{A}, \mathtt{A}, \mathtt{E}\}$ as there exist two matching graph patterns ending in `A`. Implementing non-`DISTINCT` query answering remains an open challenge.

Most neural query processors have a notion of intermediate variables and model their distribution in the entity space. For instance, having a defined distance function, geometric processors (Ren et al., 2020; Choudhary et al., 2021b) can find nearest entities as intermediate variables. Similarly, fuzzy-logic processors operating on fuzzy sets (Tang et al., 2022; Zhu et al., 2022) already maintain a scalar distribution over all entities after each execution step. Finally, GNN-based (Daza & Cochez, 2020; Alivanistos et al., 2022) and Transformer-based (Liu et al., 2022) processors explicitly include intermediate variables as nodes in the query graph (or tokens in the query sequence) and can therefore decode their representations to the entity space. The main drawback of all those methods is the lack of filtering mechanisms for the sets of intermediate variables *after* the leaf node has been identified. That is, in order to filter and project only those intermediate variables that lead to the final answer, some notion of *backward pass* is required. The first step in this direction is taken by QTO (Bai et al., 2022b) that runs the pruning backward pass after reaching the answer leaf node.

Multiple Projected Variables. The most general and complex case for queries is to have multiple projected variables as illustrated in Fig. 23 (right). In SPARQL, all projected variables are specified in the `SELECT` clause (with the possibility to project all variables in the query via `SELECT *`). In the logical form, a query has several target variables $q = ?v_1, ?v_2, ?v : \mathit{student}(\mathtt{Stanford}, ?v_1) \wedge \mathit{roommate}(?v_1, ?v) \wedge \mathit{student}(\mathtt{Stanford}, ?v_2) \wedge \mathit{classmate}(?v_2, ?v)$ such that the output bindings are organized in *tuples*. For example, one possible answer tuple is $\{?v_1 : \mathtt{F}, ?v_2 : \mathtt{F}, ?v : \mathtt{A}\}$ denotes particular nodes (variable bindings) that satisfy the query pattern.

As shown in the previous paragraph about one-variable queries, some neural query processors have the means to keep track of the intermediate variables. However, none of them have the means to construct answer tuples with variables bindings and it remains an open challenge how to incorporate multiple projected variables into such processors. Furthermore, some common caveats to be taken into account include (1) dealing with

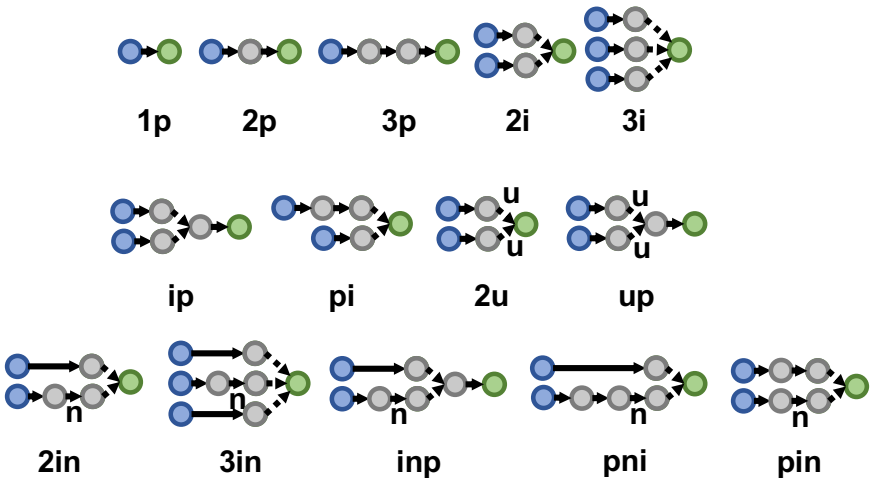

Figure 24: Standard query patterns with names, where p is projection, i is intersection, u is union, n is negation. In a pattern, blue node represents a non-variable entity, grey node represents a variable node, and the green node represents the answer node. In a typical training protocol, models are trained on 10 patterns (first and third rows) and evaluated on all patterns. In the hardest generalization case, models are only trained on *1p* queries. Some datasets further modify the patterns with additional features like qualifiers or temporal timestamps.

unbound variables that often emerge, for example, in `OPTIONAL` queries covered in Section 6.1, where answer tuples might contain an empty value ($\emptyset$ or `NULL`) for some variables; (2) the growing complexity issue where the answer set might potentially be polynomially large depending on the number of projected variables.

7 Datasets and Metrics

7.1 Evaluation Setup

Multiple datasets have been proposed for evaluation of query reasoning models. Here we introduce the common setup for CLQA task. Given a knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$, the standard practice is to split $\mathcal{G}$ into a training graph $\mathcal{G}_{train}$, a validation graph $\mathcal{G}_{val}$ and a test graph $\mathcal{G}_{test}$ (simulating the unobserved complete graph $\hat{\mathcal{G}}$ from Section 2). The standard experiment protocol is to train a query reasoning model only on the training graph $\mathcal{G}_{train}$, and evaluate the model on answering queries over the validation graph $\mathcal{G}_{val}$ and the test graph $\mathcal{G}_{test}$. Given a query q, denote the answers of this query on training, validation and test graph as $[\![q]\!]_{\texttt{train}}$, $[\![q]\!]_{\texttt{val}}$ and $[\![q]\!]_{\texttt{test}}$. During evaluation, queries may have missing answers, *e.g.*, a validation query q may have answers $[\![q]\!]_{\texttt{val}}$ that are not in $[\![q]\!]_{\texttt{train}}$, a test query q may have answers $[\![q]\!]_{\texttt{test}}$ that are not in $[\![q]\!]_{\texttt{val}}$. The overall goal of CLQA task is to find these missing answers. The details of typical training queries, training protocol, inference and evaluation metrics are introduced in Section 7.2, Section 7.3, Section 7.4, and Section 7.5, respectively.

7.2 Query Types

The standard set of graph queries used in many datasets includes 14 types: *1p/2p/3p/2i/3i/ip/pi/2u/up/2in/3in/inp/pni/pin* where p denotes relation projection, i is intersection, u is union, n is negation, and a number denotes the number of hops for projection queries or number of branches to be merged by a logical operator. Fig. 24 illustrates common query patterns. For example, *3p* is a chain-like query of three consecutive relation projections, *2i* is an intersection of two relation projections, *3in* is an intersection of three relation projections where one of the branches contains negation, *up* is a union of two relation projections followed by another projection. The original GQE by Hamilton et al. (2018)

introduced 7 query patterns with projection and intersection *1p/2p/3p/2i/3i/ip/pi*, Query2Box (Ren et al., 2020) added union queries *2u/up*, and BetaE (Ren & Leskovec, 2020) added five types with negation.

Subsequent works modified the standard set of query types in several ways, *e.g.*, hyper-relational queries (Alivanistos et al., 2022; Luo et al., 2023) with entity-relation qualifiers on relation projections, or temporal operators on edges (Lin et al., 2022a). New query patterns include queries with regular expressions of relations (property paths) (Adlakha et al., 2021), more tree-like queries (Kotnis et al., 2021), and more combinations of projections, intersections, and unions (Wang et al., 2021; Pflueger et al., 2022). We summarize existing query answering datasets and their properties in Table 15 covering supported query operators, inference setups, and additional features like temporal timestamps, class hierarchies, or complex ontological axioms.

Commonly, query datasets are sampled from different KGs to study model performance under different graph distributions, for example, BetaE datasets include sets of queries from denser Freebase (Bollacker et al., 2008) with average node degree of 18 and sparser WordNet (Miller, 1998) and NELL (Mitchell et al., 2015) with average node degree of 2. Hyper-relational datasets WD50K (Alivanistos et al., 2022) and WD50K-NFOL (Luo et al., 2023) were sampled from Wikidata (Vrandecic & Krötzsch, 2014) where qualifiers are natural. TAR datasets with class hierarchy (Tang et al., 2022) were sampled from YAGO 4 (Pellissier Tanon et al., 2020) and DBpedia (Lehmann et al., 2015) where class hierarchies are well-curated. Q2B Onto datasets with ontological axioms (Andresel et al., 2021) were sampled from LUBM (Guo et al., 2005) and NELL. Temporal TFLEX datasets (Lin et al., 2022a) were sampled from ICEWS (Boschee et al., 2015) and GDELT (Leetaru & Schrodt, 2013) that maintain event information. InductiveQE datasets (Galkin et al., 2022b) were sampled from Freebase and Wikidata, while inductive GNNQ datasets (Pflueger et al., 2022) were sampled from the WatDiv benchmark (Aluç et al., 2014) and Freebase.

7.3 Training

Query reasoning methods are trained on the given $\mathcal{G}_{train}$ with different objectives/losses and different datasets. Following the standard protocol, methods are trained on 10 query patterns *1p/2p/3p/2i/3i/2in/3in/inp/pni/pin* and evaluated on all 14 patterns including generalization to unseen *ip/pi/2u/up* patterns. That is, the training protocol assumes that models trained on atomic logical operators would learn to compositionally generalize to patterns using several operators such as *ip* and *pi* queries that use both intersection and projection.

We summarize different training objectives in Table 14. Most methods that learn a representation of the queries and entities on the graph optimize a contrastive loss, *i.e.*, minimizing the distance between the representation of a query q and its positive answers e while maximizing that between the representation of a query and negative answers e'. Various objectives include: (1) max-margin loss (first column in Table 14) with the goal that the distance of negative answers should be larger than that of positive answers at least by the margin γ. Such loss is often of the form as the equation below.

$$\ell = \max(0, \gamma - \texttt{dist}(\mathbf{q}, \mathbf{e}) + \texttt{dist}(\mathbf{q}, \mathbf{e}'));$$

(2) LogSigmoid loss (second column in Table 14) with a similar goal that pushes the distance of negatives up and vice versa. Often the loss also includes a margin term and the gradient will gradually decrease when the margin is satisfied.

$$\ell = -\log \sigma(\gamma - \texttt{dist}(\mathbf{q}, \mathbf{e})) - \sum \frac{1}{k} \log \sigma(\texttt{dist}(\mathbf{q}, \mathbf{e}') - \gamma),$$

where k is the number of negative answers. Other methods (third column in Table 14) that directly model a logit vector over all the nodes on the graph may optimize a cross entropy loss instead of a contrastive loss. Besides, methods such as the two variants of CQD (Arakelyan et al., 2021) or QTO (Bai et al., 2022b) only optimize the link prediction loss since they do not learn a representation of the query.

Almost all the datasets including GQE (Hamilton et al., 2018), Q2B (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), RegEx (Adlakha et al., 2021), BiQE (Kotnis et al., 2021), Query2Onto (Andresel et al., 2021), TAR (Tang et al., 2022), StarQE (Alivanistos et al., 2022), GNNQ (Pflueger et al., 2022), TeMP (Hu et al., 2022), TFLEX (Lin et al., 2022a), InductiveQE (Galkin et al., 2022b), SQE (Bai et al., 2023) provide

Table 14: Complex Query Answering approaches categorized under *Loss*.

Max Margin	LogSigmoid (Sun et al., 2019)	Cross Entropy
GQE (Hamilton et al., 2018), GQE w hash (Wang et al., 2019), CGA (Mai et al., 2019), MPQE (Daza & Cochez, 2020), HyPE (Choudhary et al., 2021b), Shv (Gebhart et al., 2021)	Query2Box (Ren et al., 2020), BetaE (Ren & Leskovec, 2020), RotatE-Box (Adlakha et al., 2021), ConE (Zhang et al., 2021b), NewLook (Liu et al., 2021), Q2B Onto (Andresel et al., 2021), PERM (Choudhary et al., 2021a), LogicE (Luus et al., 2021), MLPMix (Amayuelas et al., 2022), FuzzQE (Chen et al., 2022b), FLEX (Lin et al., 2022b), TFLEX (Lin et al., 2022a), SMORE (Ren et al., 2022), LinE (Huang et al., 2022b), GammaE (Yang et al., 2022a), NMP-QEM (Long et al., 2022), ENeSy (Xu et al., 2022), RoMA (Xi et al., 2022), SignalE (Wang et al., 2022a), Query2Geom (Sardina et al., 2023)	BiQE (Kotnis et al., 2021), NodePiece-QE (Galkin et al., 2022b), GNN-QE (Zhu et al., 2022), GNNQ (Pflueger et al., 2022), KGTrans (Liu et al., 2022), Query2Particles (Bai et al., 2022a), EmQL (Sun et al., 2020), LMPNN (Wang et al., 2023c), StarQE (Alivanistos et al., 2022), NQE (Luo et al., 2023), CQD$^{\mathcal{A}}$ (Arakelyan et al., 2023), SQE (Bai et al., 2023)

Table 15: Existing logical query answering datasets classified along supported **query operators**, **inference** scenarios, and **domain** properties.

		Query Operators					**Inference**		**Domain**					
Source	Dataset	Conjunctive	Union	Negation	Kleene Plus	Filter + Agg	Transductive	Inductive	Discrete	+ Timestamps	+ Continuous	Types	Rules	Qualifiers
Hamilton et al. (2018)	GQE datasets	✓					✓		✓					
Ren et al. (2020)	Q2B datasets	✓	✓				✓		✓					
Ren & Leskovec (2020)	BetaE datasets	✓	✓	✓			✓		✓					
Adlakha et al. (2021)	Regex queries	✓	✓		✓		✓		✓					
Kotnis et al. (2021)	DAG queries	✓					✓		✓					
Wang et al. (2021)	EFO-1 queries	✓	✓	✓			✓		✓					
Andresel et al. (2021)	LUBM/NELL (type)	✓	✓				✓		✓				✓	
Tang et al. (2022)	TAR datasets	✓	✓				✓		✓			✓		
Ren et al. (2022)	SMORE datasets	✓	✓				✓		✓					
Alivanistos et al. (2022)	WD50K dataset	✓					✓		✓					✓
Hu et al. (2022)	TeMP dataset	✓	✓					✓	✓			✓		
Pflueger et al. (2022)	GNNQ dataset	✓						✓	✓					
Lin et al. (2022a)	TFLEX dataset	✓	✓	✓			✓		✓	✓				
Galkin et al. (2022b)	InductiveQE dataset	✓	✓	✓				✓	✓					
Luo et al. (2023)	WD50K-NFOL dataset	✓	✓	✓			✓		✓					✓
Bai et al. (2023)	SQE dataset	✓	✓	✓			✓		✓					

a set of training queries of given structures sampled from the $\mathcal{G}_{train}$. The benefit is that during training, methods do not need to sample queries online. However, it often means that only a portion of information is utilized from the $\mathcal{G}_{train}$ since exponentially more multi-hop queries exist on $\mathcal{G}_{train}$ and the dataset can never pre-generate all offline. SMORE (Ren et al., 2022) proposes a bidirectional online query sampler such that methods can directly do online sampling efficiently without the need to pre-generate a training set offline. Alternatively, methods that do not have parameterized ways to handle logical operations, *e.g.*, CQD (Arakelyan et al., 2021), only require one-hop edges to train the overall system.

7.4 Inference

By *Inference* we understand testing scenarios on which a trained query answering model will be deployed and evaluated. Following the literature, we distinguish *Transductive* and *Inductive* inference (Fig. 25). In

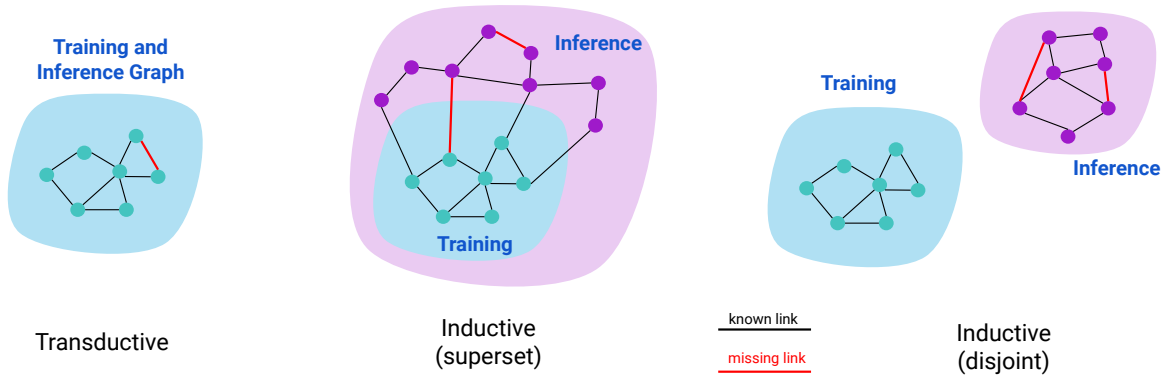

Figure 25: Inference Scenarios. In the *Transductive* case, training and inference graphs are the same and share the same nodes ($\mathcal{E}_{inf} = \mathcal{E}_{train}$). *Inductive* cases can be split into *superset* where the inference graph extends the training one ($\mathcal{E}_{train} \subseteq \mathcal{E}_{inf}$, missing links cover both seen and unseen nodes) and *disjoint* where the inference graph is disconnected ($\mathcal{E}_{train} \cap \mathcal{E}_{inf} = \emptyset$, missing links are among unseen nodes).

the transductive case, inference is performed on the graph with the same set of nodes and relation types as in training but with different edges. Any other scenario when either the number of nodes or relation types of an inference graph is different from that of the training is deemed inductive. The inference scenario plays a major role in designing query answering models, that is, transductive models can learn a shallow entity embedding matrix thanks to the fixed entity set whereas inductive models have to rely on other *invariances* available in the underlying graph in order to generalize to unseen entity/relation types. We discuss many transductive and inductive models in Section 5.2. Below, we categorize existing datasets from the *Inferece* perspective. The overview of existing CLQA datasets is presented in Table 15 through the lens of supported query operators, inference scenario, graph domain, and other features like types or qualifiers.

Transductive Inference. Formally, given a training graph $\mathcal{G}_{train} = (\mathcal{E}_{train}, \mathcal{R}_{train}, \mathcal{S}_{train})$, the transductive inference graph $\mathcal{G}_{inf}$ [9] contains the same set of entities and relation types, that is, $\mathcal{E}_{train} = \mathcal{E}_{inf}$ and $\mathcal{R}_{train} = \mathcal{R}_{inf}$, while the edge set on $\mathcal{G}_{train}$ is a subset of that on the inference graph $\mathcal{G}_{inf}$, *i.e.*, $\mathcal{S}_{train} \subset \mathcal{S}_{inf}$. In this setup, query answering is performed on the same nodes and edges seen during training. From the entity set perspective, the prediction pattern is *seen-to-seen* – missing links are predicted between known entities.

Traditionally, KG link prediction focused more on the transductive task. In CLQA, therefore, the majority of existing datasets (Table 16) follow the transductive scenario. Starting from simple triple-based graphs with fixed query patterns in GQE datasets (Hamilton et al., 2018), Query2Box datasets (Ren et al., 2020), and BetaE datasets (Ren & Leskovec, 2020) that became de-facto standard benchmarks for query answering approaches, newer datasets include regex queries (Adlakha et al., 2021), wider set of query patterns (Kotnis et al., 2021; Wang et al., 2021; Bai et al., 2023), entity type information (Tang et al., 2022), ontological axioms (Andresel et al., 2021), hyper-relational queries with qualifiers (Alivanistos et al., 2022; Luo et al., 2023), temporal queries (Lin et al., 2022a), or very large graphs up to 100M nodes (Ren et al., 2022).

Inductive Inference. Formally, given a training graph $\mathcal{G}_{train} = (\mathcal{E}_{train}, \mathcal{R}_{train}, \mathcal{S}_{train})$, the inductive inference graph $\mathcal{G}_{inf} = (\mathcal{E}_{inf}, \mathcal{R}_{inf}, \mathcal{S}_{inf})$ is different from the training graph in either the entity set or the relation set or both. The nature of this difference explains several subtypes of inductive inference. First, the set of relations might or might not be shared at inference time, that is, $\mathcal{R}_{inf} \subseteq \mathcal{R}_{train}$ or $|\mathcal{R}_{inf} \setminus \mathcal{R}_{train}| > 0$. Most of the literature on inductive link prediction (Teru et al., 2020; Zhu et al., 2021; Galkin et al., 2022a) in KGs assumes the set of relations is shared whereas the setup where new relations appear at inference time is still highly non-trivial (Huang et al., 2022a; Gao et al., 2023; Chen et al., 2023).

On the other hand, the inference graph might be either a superset of the training graph after adding new nodes and edges, $\mathcal{E}_{train} \subseteq \mathcal{E}_{inf}$, or a disjoint graph with completely new entities as a disconnected component,

[9] Below we use $\mathcal{G}_{inf}$ to refer to the graphs we use during inference, it can be $\mathcal{G}_{val}$ or $\mathcal{G}_{test}$ without loss of generalization.

$\mathcal{E}_{inf} \cap \mathcal{E}_{train} = \emptyset$ as illustrated in Fig. 25. From the node set perspective, the superset inductive inference case might contain both *unseen-to-seen* and *unseen-to-unseen* missing links whereas in the disjoint inference graph only *unseen-to-unseen* links are naturally appearing.

In CLQA, inductive reasoning is still an emerging area as it has a direct impact on the space of possible variables $\mathcal{V}$, constants $\mathcal{C}$, and answers A that might now include entities unseen at training time. Several most recent works started to explore inductive query answering (Table 16). InductiveQE datasets (Galkin et al., 2022b) focus on the inductive superset case where a training graph can be extended with up to 500% new unseen nodes. Test queries start from unseen constants and answering therefore requires reasoning over both seen and unseen nodes. Similarly, training queries can have many new correct answers when answered against the extended inference graph. GNNQ datasets (Pflueger et al., 2022) focus on the disjoint inductive inference case where constants, variables, and answers all belong to a new entity set. TeMP datasets (Hu et al., 2022) focus on the disjoint inductive inference as well but offer to leverage an additional class hierarchy as a learnable *invariant*. That is, the set of classes at training and inference time does not change.

As stated in Section 3, inductive inference is crucial for NGDBs to enable running models over updatable graphs without retraining. We conjecture that inductive datasets and models are likely to be the major contribution area in the future work.

Table 16: Complex Query Answering datasets categorized under *Inference*.

Transductive	Inductive
GQE datasets (Hamilton et al., 2018)	InductiveQE datasets (Galkin et al., 2022b)
Query2Box datasets (Ren et al., 2020)	TeMP datasets (type-aware) (Hu et al., 2022)
BetaE datasets (Ren & Leskovec, 2020)	GNNQ dataset (Pflueger et al., 2022)
Regex datasets (Adlakha et al., 2021)	
BiQE dataset (Kotnis et al., 2021)	
Query2Onto datasets (Andresel et al., 2021)	
EFO-1 dataset (Wang et al., 2021)	
TAR datasets (Tang et al., 2022)	
SMORE datasets (Ren et al., 2022)	
StarQE dataset (Alivanistos et al., 2022)	
TFLEX dataset (Lin et al., 2022a)	
WD50K-NFOL dataset (Luo et al., 2023)	
SQE dataset (Bai et al., 2023)	

7.5 Metrics

Several metrics have been proposed to evaluate the performance of query reasoning models that can be broadly classified into **generalization**, **entailment**, and **query representation quality** metrics.

Generalization Metrics. Since the aim of query reasoning models is to perform reasoning over massive incomplete graphs, most metrics are designed to evaluate models' *generalization* capabilities in discovering missing answers, *i.e.*, $[\![q]\!]_{\texttt{test}} \backslash [\![q]\!]_{\texttt{val}}$ for a given test query q. As one of the first works in the field, GQE (Hamilton et al., 2018) proposes ROC-AUC and average percentile rank (APR). The idea is that for a given test query q, GQE calculates a score for all its missing answers $e \in [\![q]\!]_{\texttt{test}} \backslash [\![q]\!]_{\texttt{val}}$ and the negatives $e' \notin [\![q]\!]_{\texttt{test}}$. The model's performance is the ROC-AUC score and APR, where they rank a missing answer against at most 1000 randomly sampled negatives of the same entity type. Besides GQE, GQE+hashing (Wang et al., 2019), CGA (Mai et al., 2019) and TractOR (Friedman & Broeck, 2020) use the same evaluation metrics.

However, the above metrics do not reflect the real world setting where we often have orders of magnitude more negatives than the missing answers. Instead of ROC-AUC or APR, Query2Box (Ren et al., 2020) proposes ranking-based metrics, such as mean reciprocal rank (MRR) and hits@k. Given a test query q, for each missing answer $e \in [\![q]\!]_{\texttt{test}} \backslash [\![q]\!]_{\texttt{val}}$, we rank it against all the other negatives $e' \notin [\![q]\!]_{\texttt{test}}$. Given the ranking r, MRR is calculated as $\frac{1}{r}$ and hits@k is $1[r \leq k]$. This has been the most used metrics for the task.

Note that the final rankings are computed only for the *hard* answers that require predicting at least one missing link. Rankings for *easy* answers reachable by edge traversal are usually discarded.

Representation Quality Metrics. Besides evaluating model's capability of finding missing answers, another aspect is to evaluate the quality of the learned query representation for all models. BetaE (Ren & Leskovec, 2020) proposes to evaluate whether the learned query representation can model the cardinality of a query's answer set, and view this as a proxy of the quality of the query representation. For models with a sense of "volume" (*e.g.*, differential entropy for Beta embeddings), the goal is to measure the Spearman's rank correlation coefficient and Pearson's correlation coefficient between the "volume" of a query (calculated from the query representation) and the cardinality of the answer set. BetaE also proposed to evaluate an ability to model queries without answers using ROC-AUC.

Entailment Metrics. The other evaluation protocol is about whether a model is also able to discover the existing answers, *e.g.*, $[\![q]\!]_{\texttt{val}}$ for test queries, that does not require inferring missing links but focuses on memorizing the graph structure (*easy* answers in the common terminology). This is referred to as faithfulness (or *entailment*) in EmQL (Sun et al., 2020). Natural for database querying tasks, it is expected that query answering models first recover *easy* answers already existing in the graph (reachable by edge traversal) and then enrich the answer set with predicted *hard* answers inferred with link prediction. A natural metric is therefore an ability to rank easy answers higher than hard answers – this was studied by InductiveQE (Galkin et al., 2022b) that proposed to use ROC-AUC as the main metric for this task.

Still, we would argue that existing metrics might not fully capture the nature of neural query answering and new metrics might be needed. For example, some under-explored but potentially useful metrics include (1) studying *reasonable* answers (in between easy and hard answers) that can be deduced by symbolic reasoners using a higher-level graph schema (ontology) (Andresel et al., 2021). A caveat in computing reasonable answers is a potentially infinite processing time of symbolic reasoners that have to be limited by time or expressiveness in order to complete in a finite time. Hence, the set of reasonable answers might still be incomplete; (2) evaluation in light of the *open-world assumption* (OWA) stating that unknown triples in the graph might not necessarily be false (as postulated by the standard *closed-world assumption* used a lot in link prediction). Practically, OWA means that even the test set might be incomplete and some high-rank predictions deemed incorrect by the test set might in fact be correct in the (possibly unobservable) complete graph. Initial experiments of Yang et al. (2022b) with OWA evaluation of link prediction explain the saturation of ranking metrics (*e.g.*, MRR) on common datasets by the performance of neural link predictors able to predict the answers from the complete graph missed in the test set. For example, saturated MRR of 0.4 on the test set might correspond to MRR of 0.9 on the true complete graph. Studying OWA evaluation in the query answering task in both transductive and inductive setups is a solid avenue for future work.

8 Applications

In addition to graph database use cases, the framework of complex query answering is applied in a variety of graph-conditioned machine learning tasks. For example, SE-KGE (Mai et al., 2020) applies GQE to answer geospatial queries conditioned on numerical $\{x, y\}$ coordinates. The coordinates encoder fuses numerical representations with entity embeddings such that the prediction task is still entity ranking.

In case-based reasoning, CBR-SUBG (Das et al., 2022) is a method for question answering over KGs based on subgraph extraction and encoding. As a byproduct, CBR-SUBG is capable of answering conjunctive queries with projections and intersections. However, due to a non-standard evaluation protocol and custom synthetic dataset, its performance cannot be directly compared to CLQA models. Similarly, Wang et al. (2023b) merge a Query2Box-like model with a pre-trained language model to improve question answering performance. LEGO (Ren et al., 2021) also applies CLQA models for KG question answering. The idea is to simultaneously parse a natural language question as a query step and execute the step in the latent space with CLQA models.

LogiRec (Wu et al., 2022) frames product recommendation as a complex logical query such that source products are root nodes, combinations of multiple products form intersections, and non-similar products to

be filtered out form negations. LogiRec employs BetaE as a query engine. Similarly, Syed et al. (2022) design an explainable recommender system based on Query2Box. Given a logical query, they first use Query2Box to generate a set of candidates and rerank them using neural collaborate filtering (He et al., 2017).

9 Summary and Future Opportunities

We presented a summary of the literature of the Complex logical query answering (CLQA) task, which is at the core of our proposed neural query engine and neural graph database (NGDB) concepts. NGDBs marry the idea of neural database and graph database where we have a unique latent space for information (including entities, relations, facts). NGDBs perform query planning, execution, and result retrieval all in the latent space using graph representation learning. Such a design choice allows for a unified, more efficient and robust interface for storage and querying. We envision NGDBs to be the next generation of databases tailored for predictive queries over sheer volumes of incomplete graph-structured data.

We also proposed a deep and detailed taxonomy of methods tackling certain aspects of the envisioned neural query engine. Going forward, there is still much room to unlock the full power of NGDB by addressing the open challenges in Neural Query Engines and Neural Storage domains. Pertaining to Neural Query Engines and adhering to the taxonomy in Section 3.2, we summarize the challenges in three main areas: Graphs, Modeling, and Queries.

Along the **Graph** branch:

- **Modality:** Supporting more graph modalities: from classic triple-only graphs to hyper-relational graphs, hypergraphs, and multimodal sources combining graphs, texts, images, and more.
- **Reasoning Domain:** Supporting logical reasoning and neural query answering over temporal and continuous (*e.g.*, textual and numerical) data – literals constitute a major portion of graphs as well as relevant queries over literals.
- **Background Semantics:** Supporting complex axioms and formal semantics that encode higher-order relationships between (latent) classes of entities and their hierarchies, *e.g.*, enabling neural reasoning over description logics and OWL fragments.

In the **Modeling** branch:

- **Encoder:** Inductive encoders supporting unseen relation at inference time – this a key for (1) **updatability** of neural databases without the need of retraining; (2) enabling the **pretrain-finetune** strategy generalizing query answering to custom graphs with custom relational schema.
- **Processor:** Expressive processor networks able to effectively and efficiently execute complex query operators akin to SPARQL and Cypher operators. Improving sample efficiency of neural processors is crucial for the *training time vs quality* tradeoff, *i.e.*, reducing training time while maintaining high predictive qualities.
- **Decoder:** So far, all neural query answering decoders operate exclusively on discrete nodes. Extending the range of answers to continuous outputs is crucial for answering real-world queries.
- **Complexity:** As the main computational bottleneck of processor networks is the dimensionality of embedding space (for purely neural models) and/or the number of nodes (for neuro-symbolic), new efficient algorithms for neural logical operators and retrieval methods are the key to scaling NGDBs to billions of nodes and trillions of edges.

In **Queries**:

- **Operators:** Neuralizing more complex query operators matching the expressiveness of declarative graph query languages, e.g., supporting Kleene plus and star, property paths, filters.

- **Patterns:** Answering more complex patterns beyond tree-like queries. This includes DAGs and cyclic graphs.
- **Projected Variables:** Allowing projecting more than a final leaf node entity, that is, allowing returning intermediate variables, relations, and multiple variables organized in tuples (bindings).
- **Expressiveness:** Answering queries outside simple EPFO and EFO fragments, extending the expressiveness to the union of conjunctive queries with negations (UCQ and UCQ_{neg} and aiming for the expressiveness of database languages.

In **Datasets** and **Evaluation**:

- The need for larger and diverse **benchmarks** covering more graph modalities, more expressive query semantics, more query operators, and query patterns.
- As the existing evaluation protocol appears to be limited (focusing only on inferring *hard* answers) there is a need for a more principled **evaluation framework** and **metrics** covering various aspects of the query answering workflow.

Pertaining to the Neural Graph Storage and NGDB in general, we identify the following challenges:

- The need for a **scalable retrieval** mechanism to scale neural reasoning to graphs of billions of nodes. Retrieval is tightly connected to the *Query Processor* and its modeling priors (as shown in Section 3.1). Existing scalable ANN libraries can only work with basic L1, L2, and cosine distances that limit the space of possible processors in the neural query engine.
- Currently, all complex query datasets listed in Section 7 provide a hardcoded query execution plan that might not be optimal for neural processors. There is a need for a **neural query planner** that would transform an input query into an optimal execution sequence taking into account prediction tasks, query complexity, type of the neural processor, and configuration of the Storage layer (that can be federated as well).
- Due to encoder inductiveness and updatability without retraining, there is a need to alleviate the issues of **continual learning** (Thrun, 1995; Ring, 1998), **catastrophic forgetting** (McCloskey & Cohen, 1989), and **size generalization** when running inference on much larger graphs than training ones (Yehudai et al., 2021; Buffelli et al., 2022).

Finally, in the era of foundation models, many research works have demonstrated various capabilities hidden inside these large language models often with hundreds of billions of parameters, and most importantly how to unleash these capabilities (Wei et al., 2022; Wang et al., 2022b; Zhou et al., 2022a). We envision an important future direction that is to design a natural language interface so that we can better harness the reasoning capabilities of these large language models for the CLQA task along the directions we mentioned ahead (Drozdov et al., 2022). Besides, NGDB also provides an exciting future opportunity to improve foundation models at various stages, especially inference. Since not all the downstream tasks require a full billion-parameter model call, it's promising to research how NGDB can accelerate or compress foundation models while still keeping the emergent behaviors that are essential for these extremely large models.

References

Vaibhav Adlakha, Parth Shah, Srikanta J. Bedathur, and Mausam. Regex queries over incomplete knowledge bases. In *3rd Conference on Automated Knowledge Base Construction, AKBC*, 2021.

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. doi: 10.1109/TPAMI.2021.3124805.

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

Uri Alon, Frank Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. Neuro-symbolic language modeling with automaton-augmented retrieval. In *International Conference on Machine Learning*, pp. 468–485. PMLR, 2022.

Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of rdf data management systems. In *13th International Semantic Web Conference*, pp. 197–212. Springer, 2014.

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.

Medina Andresel, Csaba Domokos, Daria Stepanova, and Trung-Kien Tran. A neural-symbolic approach for ontology-mediated query answering. *arXiv preprint arXiv:2106.14052*, 2021.

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

Erik Arakelyan, Pasquale Minervini, and Isabelle Augenstein. Adapting neural link predictors for complex query answering. *arXiv preprint arXiv:2301.12313*, 2023.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *Journal of artificial intelligence research*, 36:1–69, 2009.

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgVHkrYDH.

Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

Jiaxin Bai, Zihao Wang, Hongming Zhang, and Yangqiu Song. Query2Particles: Knowledge graph reasoning with particle embeddings. In *Findings of the Association for Computational Linguistics: NAACL 2022*. Association for Computational Linguistics, 2022a.

Jiaxin Bai, Tianshi Zheng, and Yangqiu Song. Sequential query encoding for complex query answering on knowledge graphs. *arXiv preprint arXiv:2302.13114*, 2023.

Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. Answering complex logical queries on knowledge graphs via query computation tree optimization. *arXiv preprint arXiv:2212.09567*, 2022b.

Pablo Barceló, Egor V. Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1lZ7AEKvB.

Pablo Barceló, Mikhail Galkin, Christopher Morris, and Miguel Romero Orth. Weisfeiler and leman go relational. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=wY_IYhh6pqj.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Maciej Besta, Patrick Iff, Florian Scheidl, Kazuki Osawa, Nikoli Dryden, Michal Podstawski, Tiancheng Chen, and Torsten Hoefler. Neural graph databases. In *Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=p0sMj8oH2O.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. *arXiv preprint arXiv:2212.06094*, 2022.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, 2008.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL https://crfm.stanford.edu/assets/report.pdf.

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

Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, James Starz, and Michael Ward. ICEWS Coded Event Data. *Harvard Dataverse*, 2015. URL https://doi.org/10.7910/DVN/28075.

Dan Brickley, Ramanathan V Guha, and Brian McBride. RDF schema 1.1. *W3C recommendation*, 25: 2004–2014, 2014.

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. {TAO}: Facebook's distributed data store for the social graph. In *2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*, pp. 49–60, 2013.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Davide Buffelli, Pietro Liò, and Fabio Vandin. SizeShiftReg: a regularization method for improving size-generalization in graph neural networks. In *Advances in Neural Information Processing Systems*, 2022.

Alison Callahan, José Cruz-Toledo, Peter Ansell, and Michel Dumontier. Bio2RDF release 2: Improved coverage, interoperability and provenance of life science linked data. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph (eds.), *The Semantic Web: Semantics and Big Data*, pp. 200–212. Springer Berlin Heidelberg, 2013.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022. URL http://jmlr.org/papers/v23/20-852.html.

Payal Chandak, Kexin Huang, and Marinka Zitnik. Building a knowledge graph to enable precision medicine. *Scientific Data*, 10(1):67, 2023. URL https://doi.org/10.1038/s41597-023-01960-3.

Mingyang Chen, Wen Zhang, Yuxia Geng, Zezhong Xu, Jeff Z Pan, and Huajun Chen. Generalizing to unseen elements: A survey on knowledge extrapolation for knowledge graphs. *arXiv preprint arXiv:2302.01859*, 2023.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022a.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxjnREFwH.

Xuelu Chen, Ziniu Hu, and Yizhou Sun. Fuzzy logic based logical query answering on knowledge graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, pp. 3939–3948. AAAI Press, 2022b.

Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan Reddy. Probabilistic entity representation model for reasoning over knowledge graphs. In *Advances in Neural Information Processing Systems*, 2021a.

Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. Self-supervised hyperboloid representations from logical queries over knowledge graphs. In *Proceedings of the Web Conference 2021*, pp. 1373–1384, 2021b.

Michael Cochez. Semantic agent programming language: use and formalization. Master's thesis, University of Jyväskylä, 2012.

Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13 (6):377–387, 1970.

Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 132–141, Valencia, Spain, 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-1013.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Hannaneh Hajishirzi, Robin Jia, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs. In *International Conference on Machine Learning, ICML 2022*, Proceedings of Machine Learning Research. PMLR, 2022.

Daniel Daza and Michael Cochez. Message passing query embedding. *Proceedings of the ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

Lauren Nicole Delong, Ramon Fernández Mir, Matthew Whyte, Zonglin Ji, and Jacques D. Fleuriot. Neurosymbolic AI for reasoning on graph structures: A survey. *arXiv preprint arXiv:2302.07200*, 2023.

Xin Luna Dong. Challenges and innovations in building a product knowledge graph. In *Proceedings of the 24th ACM SIGKDD International conference on knowledge discovery & data mining*, pp. 2869–2869, 2018.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. *arXiv preprint arXiv:2209.15003*, 2022.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pp. 265–284. Springer, 2006.

Kemele M Endris, Mikhail Galkin, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, and Sören Auer. Querying interlinked data by bridging rdf molecule templates. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIX*, pp. 1–42. Springer, 2018.

Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Journal of Web Semantics*, 19:22–41, 2013.

Emma Flint. Announcing alexa entities (beta): Create more intelligent and engaging skills with easy access to alexa's knowledge, 2021. URL https://developer.amazon.com/en-US/blogs/alexa/alexa-skills-kit/2021/02/alexa-entities-beta.

Tal Friedman and Guy Broeck. Symbolic querying of vector spaces: Probabilistic databases meets relational embeddings. In *Conference on Uncertainty in Artificial Intelligence*, pp. 1268–1277. PMLR, 2020.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the International World Wide Web Conference (WWW)*, 2013.

Mikhail Galkin, Priyansh Trivedi, Gaurav Maheshwari, Ricardo Usbeck, and Jens Lehmann. Message passing for hyper-relational knowledge graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*, pp. 7346–7359. Association for Computational Linguistics, 2020.

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*, 2022a. URL https://openreview.net/forum?id=xMJWUKJnFSw.

Mikhail Galkin, Zhaocheng Zhu, Hongyu Ren, and Jian Tang. Inductive logical query answering in knowledge graphs. In *Advances in Neural Information Processing Systems*, 2022b.

Jianfei Gao, Yangze Zhou, and Bruno Ribeiro. Double permutation equivariance for knowledge graph completion. *arXiv preprint arXiv:2302.01313*, 2023.

Thomas Gebhart, Jakob Hansen, and Paul Schrater. Knowledge sheaves: A sheaf-theoretic framework for knowledge graph embedding. *arXiv preprint arXiv:2110.03789*, 2021.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/1908.10396.

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural module networks for reasoning over text. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SygWvAVFPr.

Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 318–327. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1038. URL https://aclanthology.org/D15-1038.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Ferras Hamad, Issac Liu, and Xian Xing Zhang. Food discovery with uber eats: Building a query understanding engine, 2018. URL https://www.uber.com/blog/uber-eats-query-understanding/.

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

William L Hamilton. *Graph Representation Learning*. Morgan & Claypool Publishers, 2020.

Jakob Hansen and Robert Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3:315–358, 2019.

Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. RDF-star and SPARQL-star. *Draft Community Group Report*, 2022.

Sandro Hawke, Ivan Herman, Bijan Parsia, Axel Polleres, , and Andy Seaborne. SPARQL 1.1 entailment regimes. *W3C recommendation*, 2013.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Vinh Thinh Ho, Daria Stepanova, Dragan Milchevski, Jannik Strötgen, and Gerhard Weikum. Enhancing knowledge bases with quantity facts. In *WWW '22: The ACM Web Conference 2022*, pp. 893–901. ACM, 2022.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

Zhiwei Hu, Víctor Gutiérrez-Basulto, Zhiliang Xiang, Xiaoli Li, Ru Li, and Jeff Z Pan. Type-aware embeddings for multi-hop reasoning over knowledge graphs. *arXiv preprint arXiv:2205.00782*, 2022.

Qian Huang, Hongyu Ren, and Jure Leskovec. Few-shot relational reasoning via connection subgraph pretraining. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022a. URL https://openreview.net/forum?id=LvW71lgly25.

Xingyue Huang, Miguel Romero Orth, İsmail İlkan Ceylan, and Pablo Barceló. A theory of link prediction via relational weisfeiler-leman. *arXiv preprint arXiv:2302.02209*, 2023.

Zijian Huang, Meng-Fen Chiang, and Wang-Chien Lee. Line: Logical query reasoning over hierarchical knowledge graphs. In Aidong Zhang and Huzefa Rangwala (eds.), *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 615–625. ACM, 2022b.

Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. (comet-) atomic 2020: On symbolic and neural commonsense knowledge graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 6384–6392, 2021.

Ihab F Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed Soliman. Saga: A platform for continuous construction and serving of knowledge at scale. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 2259–2272, 2022.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular norms*, volume 8. Springer Science & Business Media, 2013.

Alexander Kolesnikov, André Susano Pinto, Lucas Beyer, Xiaohua Zhai, Jeremiah J. Harmsen, and Neil Houlsby. UVim: A unified modeling approach for vision with learned guiding codes. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=lxsL16YeE2w.

Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. Answering complex queries in knowledge graphs with bidirectional sequence encoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4968–4977. AAAI Press, 2021.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pp. 2863–2872. PMLR, 2018.

Kalev Leetaru and Philip A Schrodt. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, volume 2, pp. 1–49. Citeseer, 2013.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Xueyuan Lin, Chengjin Xu, Fenglong Su, Gengxian Zhou, Tianyi Hu, Ningyuan Li, Mingzhi Sun, Haoran Luo, et al. Tflex: Temporal feature-logic embedding framework for complex reasoning over temporal knowledge graph. *arXiv preprint arXiv:2205.14307*, 2022a.

Xueyuan Lin, Gengxian Zhou, Tianyi Hu, Li Ningyuan, Mingzhi Sun, Haoran Luo, et al. Flex: Feature-logic embedding framework for complex knowledge graph reasoning. *arXiv preprint arXiv:2205.11039*, 2022b.

Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. Neural-answering logical queries on knowledge graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1087–1097, 2021.

Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2022.

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

Haoran Luo, Haihong E, Yuhao Yang, Gengxian Zhou, Yikai Guo, Tianyu Yao, Zichen Tang, Xueyuan Lin, and Kaiyang Wan. Nqe: N-ary query embedding for complex query answering over hyper-relational knowledge graphs. In *AAAI 2023*, 2023.

Zhezheng Luo, Jiayuan Mao, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. On the expressiveness and learning of relational neural networks on hypergraphs, 2022. URL https://openreview.net/forum?id=HRF6T1SsyDn.

Francois Luus, Prithviraj Sen, Pavan Kapanipathi, Ryan Riegel, Ndivhuwo Makondo, Thabang Lebese, and Alexander Gray. Logic embeddings for complex query answering. *arXiv preprint arXiv:2103.00418*, 2021.

Gengchen Mai, Krzysztof Janowicz, Bo Yan, Rui Zhu, Ling Cai, and Ni Lao. Contextual graph attention for answering logical queries over incomplete knowledge graphs. In *Proceedings of the 10th International Conference on Knowledge Capture*, pp. 171–178, 2019.

Gengchen Mai, Krzysztof Janowicz, Ling Cai, Rui Zhu, Blake Regalia, Bo Yan, Meilin Shi, and Ni Lao. Se-kge: A location-aware knowledge graph embedding model for geographic question answering and spatial semantic lifting. *Transactions in GIS*, 24(3):623–655, 2020.

Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In *Proceedings of the 17th International Semantic Web Conference (ISWC'18)*, volume 11137 of *LNCS*, pp. 376–394. Springer, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 1003–1011, 2009.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

Eduardo Mizraji. Vector logic: a natural algebraic representation of the fundamental logical gates. *Journal of Logic and Computation*, 18(1):97–121, 2008.

Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. OWL 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, 27(65):159, 2009.

Thomas Neumann and Gerhard Weikum. Rdf-3x: A risc-style engine for rdf. *Proc. VLDB Endow.*, pp. 647–659, 2008. ISSN 2150-8097. URL https://doi.org/10.14778/1453856.1453927.

Maxwell Nye, Armando Solar-Lezama, Josh Tenenbaum, and Brenden M Lake. Learning compositional rules via neural program synthesis. *Advances in Neural Information Processing Systems*, 33:10832–10842, 2020.

Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reason-able knowledge base. In *European Semantic Web Conference*, pp. 583–596. Springer, 2020.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Maximilian Pflueger, David Tena Cucala, and Egor Kostylev. GNNQ: A neuro-symbolic approach to query answering over incomplete knowledge graphs. In *International Semantic Web Conference*, 2022.

Tahereh Pourhabibi, Kok-Leong Ong, Booi H Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020.

Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. Reasoning with language model prompting: A survey. *arXiv preprint arXiv:2212.09597*, 2022.

Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. RNNLogic: Learning logic rules for reasoning on knowledge graphs. In *International Conference on Learning Representations (ICLR)*, 2021.

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

Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations*, 2020.

Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Michihiro Yasunaga, Haitian Sun, Dale Schuurmans, Jure Leskovec, and Denny Zhou. Lego: Latent execution-guided reasoning for multi-hop question answering on knowledge graphs. In *International Conference on Machine Learning*, pp. 8959–8970. PMLR, 2021.

Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. SMORE: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2022.

Rex, Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching, 2020.

Mark B Ring. Child: A first step towards continual learning. *Learning to learn*, pp. 261–292, 1998.

Tomer Sagi, Matteo Lissandrini, Torben Bach Pedersen, and Katja Hose. A design space for rdf data representations. *The VLDB Journal*, 31(2):347–373, 2022.

Jeffrey Sardina, Callie Sardina, John D Kelleher, and Declan O'Sullivan. Analysis of attention mechanisms in box-embedding systems. In *Artificial Intelligence and Cognitive Science: 30th Irish Conference, AICS 2022*, pp. 68–80. Springer, 2023.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Haitian Sun, Andrew Arnold, Tania Bedrax Weiss, Fernando Pereira, and William W Cohen. Faithful embeddings for knowledge base queries. In *Advances in Neural Information Processing Systems*, 2020.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

Muzamil Hussain Syed, Tran Quoc Bao Huy, and Sun-Tae Chung. Context-aware explainable recommendation based on domain knowledge graph. *Big Data and Cognitive Computing*, 6(1):11, 2022.

Zhenwei Tang, Shichao Pei, Xi Peng, Fuzhen Zhuang, Xiangliang Zhang, and Robert Hoehndorf. TAR: Neural logical reasoning across tbox and abox. *arXiv preprint arXiv:2205.14591*, 2022.

Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pp. 9448–9457. PMLR, 2020.

James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. Database reasoning over text. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3091–3104. Association for Computational Linguistics, 2021a.

James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. From natural language processing to neural databases. In *Proceedings of the VLDB Endowment*, volume 14, pp. 1033–1039. VLDB Endowment, 2021b.

Sebastian Thrun. A lifelong learning perspective for mobile robot control. In *Intelligent robots and systems*, pp. 201–214. Elsevier, 1995.

Yuanyuan Tian, Wen Sun, Sui Jun Tong, En Liang Xu, Mir Hamid Pirahesh, and Wei Zhao. Synergistic graph and sql analytics inside ibm db2. *Proceedings of the VLDB Endowment*, 12(12):1782–1785, 2019.

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

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103602. URL https://www.sciencedirect.com/science/article/pii/S0004370221001533.

V Vapnik. The nature of statistical learning theory: Springer science & business media. *Berlin, Germany*, 2013.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylA_C4tPr.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57 (10):78–85, 2014.

Dingmin Wang, Yeyuan Chen, and Bernardo Cuenca Grau. Efficient embeddings of logical variables for query answering over incomplete knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2023a.

Kai Wang, Chunhong Zhang, Jibin Yu, and Qi Sun. Signal embeddings for complex logical reasoning in knowledge graphs. In *International Conference on Knowledge Science, Engineering and Management*, pp. 255–267. Springer, 2022a.

Meng Wang, Haomin Shen, Sen Wang, Lina Yao, Yinlin Jiang, Guilin Qi, and Yang Chen. Learning to hash for efficient search over incomplete knowledge graphs. In *2019 IEEE International Conference on Data Mining (ICDM)*, 2019.

Siyuan Wang, Zhongyu Wei, Jiarong Xu, and Zhihao Fan. Unifying structure reasoning and language model pre-training for complex reasoning. *arXiv preprint arXiv:2301.08913*, 2023b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Zihao Wang, Hang Yin, and Yangqiu Song. Benchmarking the combinatorial generalizability of complex query answering on knowledge graphs. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Zihao Wang, Yangqiu Song, Ginny Y. Wong, and Simon See. Logical message passing networks with one-hop inference on atomic formulas. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023c. URL https://openreview.net/forum?id=SoyOsp7i_l.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Boris Weisfeiler and Andrey Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968. English translation by G. Ryabov is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *23rd International World Wide Web Conference, WWW '14*, pp. 515–526. ACM, 2014.

Longfeng Wu, Yao Zhou, and Dawei Zhou. Towards high-order complementary recommendation via logical reasoning network. *arXiv preprint arXiv:2212.04966*, 2022.

Zhaohan Xi, Ren Pang, Changjiang Li, Tianyu Du, Shouling Ji, Fenglong Ma, and Ting Wang. Reasoning over multi-view knowledge graphs. *arXiv preprint arXiv:2209.13702*, 2022.

Wenhan Xiong, Thi-Lan-Giao Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Zezhong Xu, Wen Zhang, Peng Ye, Hui Chen, and Huajun Chen. Neural-symbolic entangled framework for complex query answering. In *Advances in Neural Information Processing Systems*, 2022.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*, 2015.

Dong Yang, Peijun Qing, Yang Li, Haonan Lu, and Xiaodong Lin. Gammae: Gamma embeddings for logical queries on knowledge graphs. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*, pp. 745–760. Association for Computational Linguistics, 2022a.

Haotong Yang, Zhouchen Lin, and Muhan Zhang. Rethinking knowledge graph evaluation under the open-world assumption. In *Advances in Neural Information Processing Systems*, 2022b. URL https://openreview.net/forum?id=5xiLuNutzJG.

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*, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Bohui Zhang, Filip Ilievski, and Pedro Szekely. Enriching wikidata with linked open data. *arXiv preprint arXiv:2207.00143*, 2022a.

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 *Advances in Neural Information Processing Systems*, volume 34, pp. 9061–9073. Curran Associates, Inc., 2021a.

Wen Zhang, Jiaoyan Chen, Juan Li, Zezhong Xu, Jeff Z Pan, and Huajun Chen. Knowledge graph reasoning with logics and embeddings: Survey and perspective. *arXiv preprint arXiv:2202.07412*, 2022b.

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

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022a.

Yangze Zhou, Gitta Kutyniok, and Bruno Ribeiro. Ood link prediction generalization capabilities of message-passing gnns in larger test graphs. In *Advances in Neural Information Processing Systems*, 2022b.

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*, 34:29476–29490, 2021.

Zhaocheng Zhu, Mikhail Galkin, Zuobai Zhang, and Jian Tang. Neural-symbolic models for logical queries on knowledge graphs. In *International Conference on Machine Learning, ICML 2022*, Proceedings of Machine Learning Research. PMLR, 2022.