Deep Learning with Knowledge Graphs Using Graph Neural Networks

Shuwen Liu

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2024

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors Professor Egor V. Kostylev, Professor Bernardo Cuenca Grau, and Professor Ian Horrocks, for their invaluable guidance, unwavering support and consistent help throughout my DPhil. My thanks go, in particular, to my main supervisor, Professor Egor V. Kostylev, for providing hands-on guidance and help for every step of my research, such as finding meaningful research problems, formulating impactful ideas, drawing insights from experiment results, and giving presentations. He demonstrated a meticulous attitude towards research, dedicated a considerable amount of time and effort in it, and contributed a lot to our papers. He is an excellent role model for me in my career path. I would also like to extend my sincere thankfulness to Professor Bernardo Cuenca Grau. He always replies to my emails promptly, has meetings and gives feedback regularly for my research. I was deeply impressed by his talent and high standard for doing research, and he has always been patiently teaching me how to approach such a standard. Finally, I would like to thank Professor Ian Horrocks for his encouragement during my tough time. He has provided significant support in sourcing fundings for my DPhil, and I learnt a lot from him, especially for thinking about problems from a broader view. Thank you all. You are the best supervisors.

This work was supported by a Dphil Research Sponsorship from Samsung Research UK, and I would like to convey my heartfelt thanks to Samsung for funding my research. This work would not have been possible without their support. My thanks also go to Keble College, for offering me nice accommodations in the past four years. Many thanks to my examiners, for taking time to read and provide feedback for my thesis.

I am also grateful to all my colleagues in the KRR group, in particular Maximilian Pflüger, Dingmin Wang, Xinyue Zhang, Yuan He, Jingchuan Shi, David Tena Cucala, and Federico Igne, whose companionship made

the journey feel a bit less arduous. Special thanks to my friend, Xing Xu. We came to Oxford in the same year, and have created a lot of happy and unforgettable memories here together. I would also like to thank my bestie, Yunting Liu. In spite of the eight-hour difference in time zones, she has been steadfastly accompanying me for four years, as if time stood still.

Last, I would like to thank my parents Min and Shaoxiong for their unconditional love and support. They always let me make my own choices in every important juncture of my life, and their only concern has always been about my health and happiness. This thesis is dedicated to you.

I have been reading *The Myth of Sisyphus* by Albert Camus [21] many times during my DPhil. Life is tough, and everyone can be Sisyphus, who unceasingly rolls a rock to the top of a mountain, but the rock falls back over and over again. In research, it is almost the same. One may get stuck for a long period without progress, be questioned or rejected, and encounter failure. At this moment, I finally understand why Camus says that Sisyphus is superior to his fate when he chooses to go back down to the suffering, even if he does not know the end. The process of moving towards the unknown bravely and ceaselessly is the most precious part. Camus concludes this philosophical essay with the following sentence, and I would also like to quote it as the ending of my acknowledgements and a summary of my DPhil journey.

"One must imagine Sisyphus happy."

Abstract

Knowledge Graphs (KGs) are reshaping the paradigm of representing, organising, and utilising information about the world. They provide rich semantic information, and have emerged as a driving power of Artificial Intelligence (AI). Primarily, there are two types of important research directions for KGs: one focuses on constructing and improving the quality of KGs, and the other delves into the wide range of applications of KGs. Recent years have also witnessed the advancement of Graph Neural Networks (GNNs), which are a class of deep learning techniques applicable to the graph domain and have demonstrated promising performance in many tasks. While there have been research attempts of applying GNNs to the KG-related tasks, there still remain several open challenges with the models' function design, scalability issues, transductive nature of being limited to predicting for entities observed during training, and the quality of the benchmarks. In this thesis, towards deep learning with KGs using GNNs, we consider the tasks of inductive KG completion and inductive knowledge-enhanced recommendation in the context of the two directions, and propose novel GNN-based approaches to address the challenges. Our extensive empirical evaluation shows that our approaches outperform the state-of-the-art approaches on a collection of baselines, and can achieve efficient training and testing in practice. We also take a further step into the KG completion problem by revisiting the benchmarks in the transductive settings. In particular, we propose a new approach to generate benchmarks that can help empirically assess models' ability to capture inference patterns. Our findings highlight the gaps between theoretical and empirical results concerning such inference ability.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Knowledge Graphs (KGs) are revolutionising the way of representing, organising, and leveraging knowledge about the world. Essentially, KGs are graph-structured knowledge bases where nodes represent entities and edges represent relations between the entities. KGs are commonly represented as sets of *triples* in a standard format such as the Resource Description Framework (RDF) [87]. The idea of KGs can be traced back to the expert systems introduced in the 1960s [73, 151]. Fundamentally, an expert system consists of a knowledge base and an inference engine, and it makes decisions by reasoning over the knowledge base using the inference engine. KGs also stem from the advances of Semantic Networks, which are knowledge bases describing semantic relations between concepts and can date back to 1950s [72], and the Semantic Web, which is a web of data that are machine-readable and was first introduced by Tim Berners-Lee in 1998 [14]. The term *Knowledge Graph* was initially proposed in the 1980s by researchers in the Netherlands [40]. KGs have been put into practice since then, and there are a large number of KGs built in recent years, including domain-specific KGs such as WordNet [90], and generic KGs such as DBpedia [5], YAGO [122], and Freebase [17]. In 2012, Google introduced a knowledge base named as Knowledge Graph, which successfully improved the quality of answers of its search engine, and brought the KGs to the masses.

KGs have emerged as a driving power of Artificial Intelligence (AI), and there has been an increasing interest in the research associated with KGs. On the one hand, numerous research endeavours focus on KGs themselves, in particular the construction [35, 81], refinement [38, 103], and alignment [150, 163] of KGs. For instance, most real-life KGs are constructed manually or semi-automatically, and thus they are often highly incomplete. In Freebase, 93.8% of persons have no *place of birth*, and 78.5%

percent of persons have no *nationality*, which are a necessary attribute for a person [91]. The incompleteness can limit the quality and use of KGs, and the problem of KG completion—the problem of extending a KG with missing triples—has garnered significant research attention [33, 117]. One of the prominent paradigms of KG completion is based on the graph embedding techniques [1, 18, 63, 77, 124, 129, 149, 164], which first embed the KG into a vector space (e.g., by learning a feature vector for each entity) and then generate the predicted triples by applying a predefined scoring function to the learnt vectors. Such techniques have demonstrated remarkable performance in *transductive settings*, where missing triples are assumed to mention only constants already occurring in the incomplete KG. A key limitation of these models, however, is that they are not applicable in *inductive* settings [50, 145, 127], (which is also referred to as KG completion with *out-of-knowledge-base* constants by Hamaguchi et al. [50]), constants unseen during training. This setting is especially relevant in practice since KGs are evolving: they may be extended with triples describing new objects or integrated with external KGs. On the other hand, the research community also exhibits a keen interest in a broad range of KG applications, spanning from question answering [13, 41], dialogue systems [130, 180], to recommender systems [142, 147] and ecotoxicology systems [96]. For example, conventional recommender systems make predictions based on user-item interactions with Matrix Factorisation (MF) techniques [48, 69, 85, 113], but often suffer from sparsity and cold-start problem [3]. To address such issues, there has been a substantial body of research efforts in leveraging external KGs to help improve accuracy, diversity, and interpretability of recommendations [139, 142, 147]. As a case in point, if a user watched the film *Titanic*, we can recommend the film *the Great Gatsby* to the user since the KG provides the information that both films are starred by Leonardo DiCaprio. Similar to the KG completion approaches mentioned above, a key limitation of these recommendation approaches is that they are transductive in nature, and are not applicable to objects (KG constants, users, or items) not observed during training. All in all, KGs have become a compelling and valuable subject of research, and have provided a great avenue for explorations with various advanced techniques.

Recent years have witnessed remarkable breakthroughs in the field of AI brought by Deep Learning (DL), as demonstrated by Convolutional Neural Networks (CNNs) in Computer Vision (CV) [70, 53], and Recurrent Neural Networks (RNNs) in Natural Language Processing (NLP) [6, 126]. In general, deep learning is a class of Machine Learning (ML) approaches that rely on neural networks with multiple layers to learn the hidden features of the raw input [71]. However, conventional neural networks,

such as CNNs and RNNs, are designed for a limited class of input such as image (grids) and text (sequences), and some open challenges arise when these networks are applied to graph-structured data such as KGs. On one side, the order of the input matters a lot for CNNs and RNNs, but the graph structure does not provide a natural order for the nodes. On the other side, CNNs and RNNs are inherently biased to neglect the structure information of a graph, which is, in fact, important information in addition to the individual features of each node.

Graph Neural Networks (GNNs) are a class of neural networks generalised to the graph domain. Primarily, the core idea of GNNs is *message passing*, where each layer of the GNNs updates the feature vector of each node in the graph by first aggregating the previous vectors of the neighbouring nodes using the aggregation function, and then combining the result with the node's previous vector using the combination function. GNNs enjoy a variety of advantages compared with conventional networks on the graph structured data. For example, GNNs are invariant under isomorphisms and insensitive to identity or ordering of nodes. Moreover, the message passing paradigm can help capture the structural information of a graph such as dependency. Over the years, researchers have delved into the exploration of more efficient and expressive GNN structures, such as GCNs [67], GATs [134], and GINs [161]. These endeavours also lead to remarkable performance improvement for tasks in a wide range of fields such as physics [11, 58], chemistry [39, 44], combinatorial optimisation [65, 68], and recommender systems [131, 177]. Overall, GNNs have become a class of prominent and rapidly advancing approaches to deal with graph structured data.

Based on the aforementioned discussion, it naturally comes to mind that one can apply the GNN techniques to solve the tasks associated with KGs, a representative of graph structured data. Although there have been a number of attempts in such direction, such as GNN-based KG completion [50, 115, 127] and GNN-based KG-enhanced recommender systems [139, 142, 147], there are still some open challenges that remain to be solved.

First of all, KGs are evolving: a KG might be extended with triples about new constants (entities) or integrated with another KG. This poses a challenge to fully exploit the inductive bias of GNNs and design inductive approaches that are applicable to triples involving constants not observed during training. R-GCN [115] is an early work for GNN-based KG completion, where the node features are initialised randomly to make predictions for triples involving unseen constants. Approaches proposed by Hamaguchi et al. [50] and Wang et al. [145] are partially inductive, as they require that each unseen constant need to be connected to at least one constant observed

during training. Another common drawback of these approaches is that they all rely on a predefined scoring function to make predictions, and depend on the standalone neighbourhoods of its constants in the input KG, without considering the common part of the neighbourhood. Although GraIL system proposed by Teru et al. [127] have addressed these drawbacks with a subgraph-extraction strategy and a distance-based encoding method, to make predictions for many triples, it requires generating a subgraph to each of them. However, KGs are usually large-scale: they may contain millions or even billions of nodes. Thus, this characteristic has become a significant bottleneck of GraIL in practice. Recently, some path-based approaches using GNNs have yielded promising results, such as NBFNet [181] and PathCon [138]. In particular, NBFNet [181] learns the pairwise node representations by summarising the path representations between the node pair. We see NBFNet as a concurrent work with our project. Overall, it is still imperative to propose an effective and efficient GNN-based approach for inductive KG completion.

Furthermore, the inductive settings become more challenging for KG-enhanced recommender systems, where the interaction graph can also be extended with new items or users. On the one hand, prominent knowledge-enhanced recommender systems, such as RippleNet [139], KGCN [142], and MKR [141], are not inductive and can not generalise to unseen objects. On the other hand, state-of-the-art inductive recommender systems, such as IDCF [155] and IGMC [177], do not support KGs. Moreover, IDCF is only partially inductive and still requires retraining for unseen items, whereas IGMC suffers from the scalability issue. Therefore, it remains elusive to find a solution for inductive recommender systems that support KGs and address these limitations.

Finally, KGs are expressive and can carry rich semantic information. As a result, they can be used to assess KG completion models' ability to capture *inference patterns*—that is, an abstraction of a set of logical rules describing a type of causal dependencies that may exist in the KG [1]. Take *symmetry* as an example of such patterns: if relation 'is friend' is symmetric and a is a friend of b, then b is also a friend of a. Some theoretical studies [1, 124] have analysed the ability of a variety of KG completion systems to capture inference patterns. For instance, it has been shown that RotatE [124], a KG completion system based on graph embedding techniques, is able to capture symmetry. In particular, if a relation is defined as symmetric by a rule, there exists a configuration of the models' parameters associated to the relation. In this configuration, for any dataset, the models' predictions will coincide with the facts derived by the rule. However, none of these theoretical results have

included the GNN-based systems. Moreover, there is little empirical indication of the models' capabilities to learn relevant patterns *in practice*. Standard completion benchmarks are, nevertheless, not well-suited for evaluating models' abilities to learn patterns, because the training and test sets of these benchmarks are a random split of a given KG. In light of these limitations, there is a need for benchmarks that take into account the causal dependencies inherent to inference patterns.

This thesis aims to address the open challenges discussed above, and exploit the power of GNNs for KG-oriented tasks such as KG completion and KG-related applications such as KG-enhanced recommender systems.

1.2 Thesis Outline and Contributions

Towards exploring deep learning on KGs using GNNs, this thesis summarises our work exploiting the power of GNNs in improving the quality of KGs through KG completion, and leveraging the semantic information in KGs to provide external knowledge for applications such as recommender systems. In particular, we focus on a more challenging and practical inductive setting for both tasks. Moreover, in addition to improving the predictive power of GNN-based models, it is also important to understand why such predictions are made and what the models have learnt. Therefore, this work also includes our investigation about the KG completion models' ability to capture inference patterns from an empirical perspective. In the rest of this section, we will present an outline and the main contributions of this thesis.

In Chapter 2, we provide a recapitulation of necessary preliminaries and mathematical notations that will be broadly used throughout the thesis. In particular, we introduce the conceptual foundations underpinning KGs, KG completion, KG completion benchmarks and evaluation metrics, Datalog, and GNNs.

In Chapter 3, we propose a novel GNN-based inductive KG completion approach using pair-wise encoding, where the KG is fully encoded into a GNN in a transparent way, and where the predicted triples can be read out directly from the last layer of the GNN without the need for additional components or scoring functions. An extensive and thorough literature review of KG completion approaches is also presented in this chapter. In addition, we also provide an analysis to show that the encoding size of our approach is linear in practice. Furthermore, we highlight the insights behind the contribution of our approach with a discussion on the use of unary or binary encoders for the GNN-based KG completion approach. Moreover, we implement our approach in a system called INDIGO, and compare it with state-of-the-art KG completion

systems on the inductive benchmarks proposed in the previous papers. Based on Freebase, we also propose a new inductive KG completion benchmark called INDIGO-BM, where the use of unseen constants in the test sets is not restricted in any specific way, and type information is also included. We show in our experiments that both on the existing benchmarks and the new benchmark we proposed, INDIGO demonstrates superior performance over the baselines, and it can also be trained and tested more efficiently. Furthermore, we exploit the impact on the system's performance with different GNN variants using an ablation study, and show that GCN is an effective choice. Finally, we identify a set of high-confidence rules represented by the inference patterns over the types and relations in the KG, and show in practice that INDIGO outperforms the other systems in capturing these logic rules. Note that here the definition of capturing logic rules is different from those in the theoretic studies [1, 124] discussed in Section 1.1, and we will elaborate the difference in Section 3.3.4 and Section 5.1. Moreover, we find that the number of high-confidence rules is often very limited in the existing benchmarks, and we will address this limitation in Chapter 5.

In Chapter 4, we further apply our neural architecture for inductive KG completion proposed in Chapter 3 to the knowledge-enhanced recommendation task. In particular, we present a new recommendation approach which is effective and scalable in inductive settings and can seamlessly incorporate KGs. Along with a comprehensive investigation of related work about GNN-based recommender systems and knowledge-enhanced recommender systems, to the best of our knowledge, our approach is the first to exploit the power of GNNs to address the inductive recommendation problem while incorporating KGs. We implement our approach in a system called InKER, and show in experiments that our system, while being more general than the state-of-the-art baselines, outperforms them in the restricted cases where the baselines are applicable. Last but not least, we also present an ablation study investigating the impact on system's performance of using different variants of GNNs, and an analysis about the significance of incorporating KGs when the rating matrix is sparse.

In Chapter 5, we explore more about the KG completion systems' ability to capture inference patterns, and we consider the transductive setting for simplicity. As mentioned above, in Chapter 3, we find that there are a very limited number of high-confidence rules in the existing KG completion benchmarks. To address this limitation, in this chapter, we propose theoretical requirements for inferential KG completion benchmarks: there is a set of logical rules so that the missing facts are the results of the rules' application; the training set includes both premises matching

rule antecedents and the corresponding conclusions; the test set consists of the results of applying the rules to the training set; and the negative examples are designed to discourage the models from learning rules not entailed by the rule set. Following the requirements, we provide an in-depth analysis about existing KG completion benchmarks, showing that none of them satisfy all our requirements for inferential benchmarks. Furthermore, we propose a pioneering approach for designing inferential KG completion benchmarks satisfying these requirements, where the models are exposed during training to both premise and conclusion triples for selected rules, the triples in the test set have supporting evidence in the training set, and the models are penalised for learning unintended rules. By applying our approach, we present a collection of inferential benchmarks, called LogInfer, based on common inference patterns and the KGs underpinning subsets of three real-world KGs. We also present a comprehensive evaluation of embedding-based, GNN-based and rule mining KG completion systems on these benchmarks. Through the empirical study, we find that all systems performed significantly worse on our benchmarks than on the standard benchmarks; rule-based systems outperformed others on simple inference patterns, but not on complex patterns; some models achieved favourable performance although they are shown in theory that they can not capture certain patterns; performance on classification metrics relies heavily on the negative sampling strategy, and our methods are able to generate more challenging negative examples. These observations highlight the advantages of our approach, and provide interesting insights for the future improvement of KG completion approaches.

In Chapter 6, we summarise this thesis with a brief review about our contributions and a discussion about future work.

1.3 Publications

This thesis is based on my publications listed below [78, 79]:

1. Shuwen Liu, Bernardo Cuenca Grau, Ian Horrocks, Egor V. Kostylev. INDIGO: GNN-based inductive knowledge graph completion using pair-wise encoding. In *NeurIPS*, pages 2034–2045, 2021.

2. Shuwen Liu, Bernardo Cuenca Grau, Ian Horrocks, and Egor V. Kostylev. Revisiting inferential benchmarks for knowledge graph completion. In *KR*, pages 461–471, 2023.

Chapter 2

Preliminaries

In this chapter, we recapitulate the basic notions that will be broadly used throughout the thesis. In what follows, we first introduce the fundamental concepts underpinning KGs and give a definition to the KG completion problem in Section 2.1. Then, in Section 2.2 and 2.3, we briefly introduce the standard approaches to KG completion benchmarking and the evaluation metrics they employ. Furthermore, in Section 2.4, we overview the basics of Datalog and inference patterns that are related to this thesis. In Section 2.5, we introduce in a concise manner of GNNs, and present an overview of some prominent structures of them. Additional definitions will also be provided in subsequent chapters if necessary. Finally, we list the necessary mathematical notations and examples of symbols in Section 2.6.

2.1 KGs and KG Completion

In the context of this thesis, a *signature* consists of pairwise disjoint sets of *types* and *relations*, which are often collectively referred to as *predicates*, and *constants*, which are also referred to as *entities*.

A *knowledge graph* (KG) is a finite set of triples of the form (e, type, t), where e is a constant and t is a type, and of the form (s, R, o), where s, o are constants and R is a relation. For $\mathcal{K}$ a KG, let $\mathsf{Sig}(\mathcal{K})$, $\mathsf{Types}(\mathcal{K})$, $\mathsf{Rels}(\mathcal{K})$, $\mathsf{Preds}(\mathcal{K})$, and $\mathsf{Consts}(\mathcal{K})$ be the signature, the (set of) types, relations, predicates, and constants used in $\mathcal{K}$, respectively. We write $\mathsf{Sig}(\mathcal{K}) \subseteq \mathsf{Sig}(\mathcal{K}')$ if the signature of $\mathcal{K}$ only uses constants and predicates in the signature of $\mathcal{K}'$.

In some of the KG literature, they do not distinguish the two forms of KG triples mentioned above. In other words, constants and types are sometimes collectively referred to as *entities*, and the type keyword is consequently treated as a normal relation.

The concept of KGs is also closely related to the Web Ontology Language (OWL) [57, 56], which is not included in the scope of this thesis. We refer the readers to for the OWL-based KGs [29, 55, 152].

A *KG completion* problem can be intuitively understood as extending a KG $\mathcal{K}$ to its complete version $\mathcal{K}^*$ by adding triples that are over the same signature. Here, we first present the formal definition of *inductive* KG completion, and then define *transductive* KG completion, which can be seen as a particular case of the inductive version.

Definition 1. *Given arbitrary (but fixed) finite sets* Types *and* Rels *of types and relations, the goal of* inductive *KG completion is to learn a Boolean* completion function $f(\cdot,\cdot)$ *that is applicable to each pair of a KG* $\mathcal{K}$ *with* $\mathsf{Pred}(\mathcal{K}) \subseteq \mathsf{Types} \cup \mathsf{Rels}$ *and triple* λ *with* $\mathsf{Sig}(\lambda) \subseteq \mathsf{Sig}(\mathcal{K})$ *such that* $f(\mathcal{K}, \lambda)$ *is true if* λ *is in* $\mathcal{K}^*$*. The task of* transductive *KG completion is to learn a Boolean completion function* $f_{\mathcal{K}}(\cdot)$ *applicable to triples over* $\mathsf{Sig}(\mathcal{K})$ *such that* $f_{\mathcal{K}}(\lambda)$ *is true if* λ *is in* $\mathcal{K}^*$*.*

The confidence-based variant of these two tasks assumes that $f(\mathcal{K}, \lambda)$ (or $f_{\mathcal{K}}(\lambda)$) is, for each triple λ, a value in $[0, 1]$ representing the likelihood that λ holds in $\mathcal{K}^*$.

Figure 2.1 depicts an example for KG completion. If we use the graph enclosed within the frame for training, both transductive and inductive KG completion functions will be able to predict the missing triple (*Plato*, lives, *Greece*). Now, assume the graph is now extended with the triples

$$(\mathit{Aristotle}, \mathsf{student}, \mathit{Plato}), (\mathit{R.Feynman}, \mathsf{student}, \mathit{J.Wheeler}), \quad (\mathit{J.Wheeler}, \mathsf{lives}, \mathit{USA}),$$

introducing constants for Aristotle, Feynman, Wheeler, and the USA. The inductive functions can take such new information into account, and predict the missing triples

$$(\mathit{Aristotle}, \mathsf{lives}, \mathit{Greece}), (\mathit{R.Feynman}, \mathsf{lives}, \mathit{USA}).$$

In contrast, the transductive functions will not be applicable to make such predictions for the new constants.

2.2 KG Completion Benchmarks

In this section, we first introduce the components of KG completion benchmarks, then describe the most commonly used strategy of generating negative examples for such benchmarks, and finally recapitulate a series of standard metrics for KG completion evaluation.

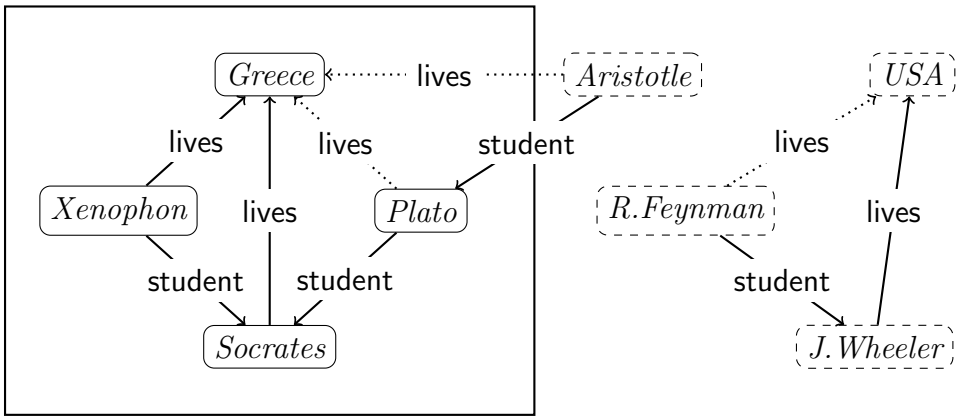


Figure 2.1: An example for KG completion.

Benchmarks play a significant role in evaluating KG completion methods. They can help reveal the actual progress of KG completion techniques, identify potential issues in current approaches, and motivate further improvement in this field. Transductive KG completion benchmarks usually contain disjoint sets $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$ (possibly), and $\mathcal{P}_{\text{test}}$ of triples for training, validation, and testing, respectively, with $\mathsf{Sig}(\mathcal{P}_{\text{valid}}) \subseteq \mathsf{Sig}(\mathcal{P}_{\text{train}})$ and $\mathsf{Sig}(\mathcal{P}_{\text{test}}) \subseteq \mathsf{Sig}(\mathcal{P}_{\text{train}})$. Thus, the signature of $\mathcal{P}_{\text{valid}}$ and the signature of $\mathcal{P}_{\text{test}}$ only uses constants and predicates in the signature of $\mathcal{P}_{\text{train}}$. Moreover, let P_{test} be the set of positive testing examples in the benchmark, and P_{test} contains all triples λ from $\mathcal{P}_{\text{test}}$ (which is analogous to both training and validation). In contrast, inductive KG completion benchmarks usually contain disjoint sets $\mathcal{P}_{\text{train}}$ and $\mathcal{P}_{\text{valid}}$ of triples with $\mathsf{Sig}(\mathcal{P}_{\text{valid}}) \subseteq \mathsf{Sig}(\mathcal{P}_{\text{train}})$ for training and validation, an incomplete testing KG $\mathcal{K}_{\text{test}}$, and a set $\mathcal{P}_{\text{test}}$ of triples with $\mathsf{Sig}(\mathcal{P}_{\text{test}}) \subseteq \mathsf{Sig}(\mathcal{K}_{\text{test}})$ that are to hold in the completion of $\mathcal{K}_{\text{test}}$ for testing. Here, the set P_{test} of positive testing examples contains all possible pairs $(\mathcal{K}_{\text{test}}, \lambda)$ of the incomplete testing KG $\mathcal{K}_{\text{test}}$, and a triple λ from $\mathcal{P}_{\text{test}}$. Let $\mathcal{P}_{\text{all}}$ denote the union of all the sets of triples in a benchmark. We have $\mathcal{P}_{\text{all}} = \mathcal{P}_{\text{train}} \cup \mathcal{P}_{\text{valid}} \cup \mathcal{P}_{\text{test}}$ for transductive benchmarks, and $\mathcal{P}_{\text{all}} = \mathcal{P}_{\text{train}} \cup \mathcal{P}_{\text{valid}} \cup \mathcal{K}_{\text{test}} \cup \mathcal{P}_{\text{test}}$ for inductive benchmarks. In addition to positive examples for training, validation, and testing, each KG completion benchmark may also contain sets N_{train}, N_{valid}, and N_{test} of negative examples for training, validation, and testing, respectively [23, 112]. However, in existing benchmarks, it is more common not to include explicit negative examples, but instead assume a sampling strategy to generate such negative examples [120], and adopt a *(partial) closed-world* assumption that all the triples not in $\mathcal{P}_{\text{all}}$ are false. A straightforward strategy is to take all the unobserved triples (paired with the KG for the inductive settings) as negative

examples. Nevertheless, this may lead to a significant and unacceptable imbalance between the number of positive and negative examples. Moreover, evaluating on such a huge number of negative examples is computationally prohibitive.

The most commonly used negative sampling strategy is *random corruption*. In particular, to generate a negative example, they randomly *corrupt* one of the three components in each positive example triple [18]. For instance, for a positive test example (s, R, o) in a transductive benchmark, the triples (s', R, o), (s, R', o), and (s, R, o') may be taken as negative examples, where s' and o' are randomly sampled from $\mathsf{Consts}(\mathcal{P}_{\text{train}})$, R' is randomly sampled from $\mathsf{Rels}(\mathcal{P}_{\text{train}})$, so that the resulting triple is not included in $\mathcal{P}_{\text{all}}$. Note that it is not necessary to generate with such three types of corruptions simultaneously. For example, in Bordes et al. [18], they only corrupt s and o, and the relation R is retained. However, it is empirically shown that the negative examples generated by random corruption can be very easily predicted as false, and even lead to nearly perfect performance among most of the systems [7, 23, 112].

2.3 Evaluation Metrics of KG Completion

In general, KG completion systems are evaluated on benchmarks using *classification-based* metrics and *ranking-based* metrics, which are computed based on their predictions on sets P_{test} and N_{test} of positive and negative examples.

2.3.1 Classification-Based Metrics

A family of basic classification-based metrics rely on the counts of *true positive*, *true negative*, *false positive*, and *false negative* predictions that are computed in the usual way from P_{test} and N_{test}, and the systems' binary prediction on P_{test} and N_{test}.

More precisely, a *true positive* prediction indicates that a system returns *positive* on a *positive* example; a *true negative* prediction means that a system returns *negative* on a *negative* example; a *false positive* prediction occurs when a system returns *positive* on a *negative* example; whereas a *false negative* prediction is a type of error when a system returns *negative* on a *positive* example. Let TP, TN, FP, and FN denote the number of true positive, true negative, false positive, and false negative predictions on P_{test} and N_{test}, respectively. We then introduce the commonly used standard classification-based metrics computed from TP, TN, FP, and FN.

Precision. *Precision* is a metric that quantifies the correctness of the systems' positive predictions. It is defined as the ratio of true positive predictions to the total

number of positive predictions made by the model, namely,

$$Precision = \frac{TP}{TP + FP}. \tag{2.1}$$

Precision is preferred when we aim to ensure that the positive predictions made by the systems are highly accurate.

Recall. *Recall*, also known as *sensitivity* or *true positive rate (TPR)*, is a metric that measures systems' capability to identify all the positive examples. It is defined as the proportion of true positive predictions to the total number of positive examples in $\mathcal{P}_{\text{test}}$, namely,

$$Recall = \frac{TP}{TP + FN}. \tag{2.2}$$

Recall is particularly valuable when it is crucial to cover as many positive examples in the predictions as possible.

F1 Score. *F1 score* is a metric that provides a summation of precision and recall for the system performance. It is defined as

$$F1\ score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \tag{2.3}$$

F1 score is especially useful when a single metric is needed to capture the trade-off between precision and recall, with a higher score indicating a better trade-off in this regard.

Accuracy. *Accuracy* is a metric that assesses the overall correctness of the system predictions. It is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.4}$$

Accuracy provides a holistic view of the systems' performance on both positive and negative examples. However, accuracy can be misleading when the benchmark is highly imbalanced. For example, consider a benchmark where there is only 1 positive example but 99 negative examples. A naive system by guessing all cases as negative will be able to achieve an accuracy 0.99. To mitigate this limitation, metrics such as precision, recall, and F1 score can be used when positive examples are significantly outnumbered by negative examples. Consider the aforementioned example, the naive system will obtain zeros for precision, recall, and F1 score on this benchmark, since TP is 0. However, these metrics are no longer as effective when the number of positive examples is much larger than the number of negative examples. For example, in a benchmark with 99 positive examples and 1 negative example, we consider another

naive system predicting all the examples as positive. Indeed, such a naive system will show strong performance on these metrics, with an accuracy of 0.99, precision of 0.99, recall of 1, and F1 score of 0.99. This shortcoming, however, can be alleviated by some other metrics such as *false positive rate*.

False Positive Rate. *False positive rate (FPR)* is a metric defined as the proportion of false positives to the total number of negative examples, namely,

$$False\ positive\ rate = \frac{FP}{FP + TN}. \tag{2.5}$$

False positive rate can help assess the ability of a system to make accurate predictions on negative examples, and a higher value indicates that the system is less effective at distinguishing negative examples. For the earlier discussed case where there are 99 positive examples but only 1 negative example, it will result in a false positive rate of 1, and thus ultimately disclose the weakness of the naive system.

True Negative Rate. *True negative rate*, known as *specificity*, is another metric measuring how well the model can identify negative examples in the dataset. It is defined as the proportion of negative examples that are correctly identified by the model, namely,

$$True\ negative\ rate = \frac{TN}{TN + FP}. \tag{2.6}$$

A high true negative rate, or specificity value, indicates that the model is good at identifying negative instances and has fewer false positives.

It is also worth noting that in addition to KG completion systems directly making binary predictions, there are also many KG completion systems with confidence predictions (such as numerical scores from 0 to 1), and they often rely on a threshold hyperparameter to transform the scores into binary predictions. Thus, we then introduce another two threshold-independent metrics that can be used to evaluate the overall performance of a system without being tied to a specific threshold: *Receiver Operating Characteristic - Area Under the Curve (ROC-AUC)* and *the Area Under the Precision-Recall Curve (AUC-PR)*.

ROC-AUC. A ROC curve is created by plotting false positive rate on the x-axis and true positive rate (recall) on the y-axis across various thresholds. *ROC-AUC* computes the area under the ROC curve, and measures the probability that a system ranks a randomly selected positive example higher than a randomly selected negative example.

AUC-PR. A PR curve is created by plotting recall on the x-axis and precision on the y-axis across various thresholds. *AUC-PR* quantifies the area under the PR curve, and

summarises the systems' overall performance on precision and recall among different thresholds.

2.3.2 Ranking-Based Metrics

The ranking-based metrics rely on confidence predictions in $[0, 1]$ and hence are applicable only to systems solving the confidence-based variant of the KG completion task. These metrics usually take into account only triples of the form (s, R, o), but a generalisation to triples (c, type, t) is straightforward. For each positive example p (i.e., a triple (s, R, o) for the transductive setting, or a pair $(\mathcal{K}_{\text{test}}, (s, R, o))$ for the inductive setting) in P_{test}, let N_p^{s} be the subset of N_{test} of all negative examples of the form (s', R, o) for transductive benchmarks, or of the form $(\mathcal{K}_{\text{test}}, (s', R, o))$ for inductive benchmarks, with $s \neq s'$, and let N_p^{R} and N_p^{o} be computed analogously. Then, for each $x \in \{\mathsf{s}, \mathsf{R}, \mathsf{o}\}$, let $rank_x(p)$ be the position of p in the ordering of $\{p\} \cup N_p^x$ based on the prediction confidences of the model (trained on P_{train}, N_{train} and validated on P_{valid}, N_{valid}). Here are the commonly used ranking-based metrics computed using $rank_x(p)$.

Hits@k. The *constant-hit* C-Hits@k and *relation-hit* R-Hits@k metrics for a number $k \in \mathbb{N}$ are defined as

$$\text{C-Hits@}k = (\text{Hits}_{\mathsf{s}}@k + \text{Hits}_{\mathsf{o}}@k)/2, \tag{2.7}$$

$$\text{R-Hits@}k = \text{Hits}_{\mathsf{R}}@k, \tag{2.8}$$

$$\text{for every } x, \text{Hits}_x@k = |\{p \in P_{\text{test}} \mid rank_x(p) \leq k\}|/|P_{\text{test}}|. \tag{2.9}$$

Hits@k measures how well a system's top-k predictions match the ground truth, but is sensitive to the choice of the value k. It does not consider the quality of predictions beyond the first k entries.

MRR. The *Mean Reciprocal Rank* (*MRR*) *for constants* and *relations* are defined as

$$\text{C-MRR} = (\text{MRR}_{\mathsf{s}} + \text{MRR}_{\mathsf{o}})/2, \tag{2.10}$$

$$\text{R-MRR} = \text{MRR}_{\mathsf{R}}, \tag{2.11}$$

$$\text{for every } x, \text{MRR}_x = \left(\sum_{p \in P_{\text{test}}} \frac{1}{rank_x(p)} \right) \Big/ |P_{\text{test}}|. \tag{2.12}$$

MRR is a commonly used metric of ranking performance, and it reflects how soon we can find the ground truth in a list of candidates. However, MRR only considers the rank of the first correct candidate, which makes it less robust to variations in the rest of the ranking list.

2.4 Datalog

Several chapters of the thesis will rely on the concept of inference patterns. The logic rules, which are instantiations of inference patterns, are often represented in Datalog [2], a well-known rule language for knowledge representation. Therefore, in this section we first review the basics of Datalog, and then provide definitions of inference patterns.

In this thesis, a (Datalog) *atom* is an expression of the form (d, type, t), (d_1, R, d_2), or $(d_1 \neq d_2)$ where t is a type, R is a relation, and each of d, d_1 and d_2 is either a constant or a *variable*. A (Datalog) *rule* is a function-free first-order logic sentence of the form

$$\forall \mathbf{x}.\ B_1 \wedge \cdots \wedge B_n \rightarrow H, \tag{2.13}$$

where H is a $\neq$-free atom called the *head* of the rule, all B_i are atoms serving together as the *body* of the rule, and each variable in the rule is in $\mathbf{x}$ and mentioned in some $\neq$-free B_i. A (Datalog) *program* is a finite set of rules.

A *substitution* is a mapping of variables to constants, and it applies atoms and conjunctions of atoms in the standard manner. Each rule r of form (2.13) realises a *one-step rule application* $\mathcal{T}_r$ on KGs: for a KG $\mathcal{K}$, KG $\mathcal{T}_r(\mathcal{K})$ consists of all triples $\sigma(H)$ for all *witnesses* of the body in $\mathcal{K}$—that is, substitutions σ such that $\sigma(d_1) \neq \sigma(d_2)$ for each B_i of the form $(d_1 \neq d_2)$ and $\sigma(B_i) \in \mathcal{K}$ for each other B_i. We call all such $\neq$-free $\sigma(B_i)$ in $\mathcal{K}$ the *premise triples* for r in $\mathcal{K}$, and all such $\sigma(H)$ the *conclusion triples* for r in $\mathcal{K}$; furthermore, we call all such σ the *support* of r in $\mathcal{K}$. Note that a triple may simultaneously function as both a premise and a conclusion triple. The *one-step application* $\mathcal{T}_\mathcal{R}(\mathcal{K})$ of a program $\mathcal{R}$ to a KG $\mathcal{K}$ is defined as $\mathcal{T}_\mathcal{R}(\mathcal{K}) = \bigcup_{r \in \mathcal{R}} \mathcal{T}_r(\mathcal{K})$.

Materialisation (or *forward chaining*) is a reasoning paradigm which consists of successive rounds of one-step rule applications until no new triples can be derived for an input program and a KG [95], which always happens due to the fixpoint semantics of Datalog. For a program $\mathcal{R}$ and a KG $\mathcal{K}$, the *materialisation* $\mathcal{M}_\mathcal{R}(\mathcal{K})$ of $\mathcal{R}$ on $\mathcal{K}$ is defined as $\mathcal{M}_\mathcal{R}(\mathcal{K}) = \bigcup_{i \geq 0} \mathcal{K}_i$, where $\mathcal{K}_0 = \mathcal{K}$ and $\mathcal{K}_{i+1} = \mathcal{T}_\mathcal{R}(\mathcal{K}_i) \cup \mathcal{K}_i$ for each $i \geq 0$. A set $\mathcal{R}$ of rules *logically entails* another set $\mathcal{R}'$, written as $\mathcal{R} \models \mathcal{R}'$ of rules, if $\mathcal{M}_{\mathcal{R}'}(\mathcal{K}) \subseteq \mathcal{M}_\mathcal{R}(\mathcal{K})$ for each KG $\mathcal{K}$.

An *inference (rule) pattern* is an expression of the form

$$\mathfrak{B}_1 \wedge \cdots \wedge \mathfrak{B}_n \rightarrow \mathfrak{H}, \tag{2.14}$$

where the $\mathfrak{B}_i$ and $\mathfrak{H}$ are same as the B_i and H in rule (2.13), respectively, except that instead of types t and relations R they use *type templates* $\_\mathsf{t}$ and *relation templates*

_R, which are originating from dedicated infinite sets of symbols. It should also be noted that the template in $\mathfrak{H}$ may or may not be mentioned in one or several $\mathfrak{B}_i$. The expressions of the form (2.14) are also called *rule templates*. Examples of inference patterns include *symmetry* (e.g., if relation 'is colleague' is symmetric and a is a colleague of b, then b is also a colleague of a); *inversion* (e.g., if relation 'is parent' is the inversion of the relation 'is child', and a is a parent of b, then b is a child of a); *composition* (e.g., if 'is grandmother of' is the composition of the relation 'is mother of' with itself, a is the mother of b, and b is the mother of c, then a is a grandmother of c); and *intersection* (e.g., if 'is mother of' is the intersection of relations 'is parent of' and 'gives birth to', a is a parent of b, and a gives birth to b, then a is the mother of b). Consider this intersection pattern, it can be expressed using the following rule template, where _R, _S, and _T can be instantiated to arbitrary relations:

$$(x, \_\mathsf{R}, y) \wedge (x, \_\mathsf{S}, y) \rightarrow (x, \_\mathsf{T}, y). \tag{2.15}$$

A rule r is *represented* by a inference pattern $\mathfrak{p}$ if r can be obtained from $\mathfrak{p}$ by substituting the type and relation templates in $\mathfrak{p}$ by types and relations, respectively. For instance, our example (2.15) of the intersection pattern involving parenthood relations can be instantiated as the following rule:

$$(x, \mathsf{IsParent}, y) \wedge (x, \mathsf{GivesBirth}, y) \rightarrow (x, \mathsf{IsMother}, y). \tag{2.16}$$

2.5 Graph Neural Networks

GNNs are a class of deep learning architecture applicable to graph-structured data. Before defining GNNs, we first define the node-annotated graphs, which are the graphs manipulated by GNN.

Definition 2. *For* annotation dimension $\delta \in \mathbb{N}$*, a* δ-annotated graph *is an undirected graph* $G = (V, E)$ *where each node* $v \in V$ *is associated with a* feature *vector* $\mathbf{v} \in \mathbb{R}^\delta$*. The* i*-th element of* $\mathbf{v}$ *is denoted by* $(\mathbf{v})_i$*.*

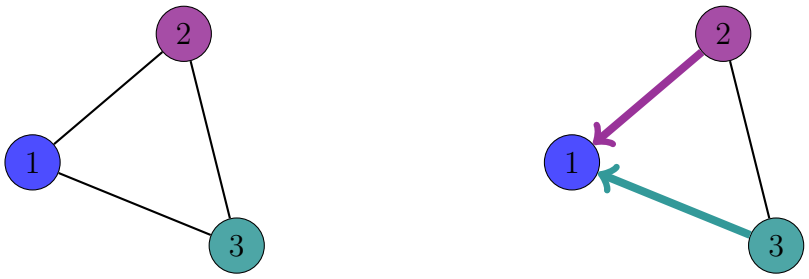
Note that the graph is assumed to be undirected in most of the cases, but in some recent work the graph can also be directed [110]. The feature vectors are often generated from one-hot encoding of the current node's index, randomly initialised, or using auxiliary attribute information. Generally speaking, a GNN takes a δ-annotated graph G as input, and updates the feature vector of each node in the graph iteratively through (multiple) layers.

The essence of GNNs lies in how to update and learn the representations (feature vectors) of nodes, and a defining feature of the approaches is known as *message passing*. In particular, each GNN layer updates the feature vector of a node by first aggregating the previous vectors of the neighbouring nodes using the aggregation function, and then combining the result with the node's previous vector using the combination function. Formally, a L-layer GNN updates, on each layer $\ell \in \{1, \ldots, L\}$, the vector of node v using a message passing rule of the following form, which first aggregates the vectors of the neighbouring nodes of v in layer $\ell-1$, and then combines the results with $\mathbf{v}^{\ell-1}$:

$$\mathbf{v}^{\ell} = \textsc{Combine}^{\ell}\left(\mathbf{v}^{\ell-1}, \textsc{Aggregate}^{\ell}\left(\{\{(\mathbf{v}')^{\ell-1} \mid v' \in \mathcal{N}(v)\}\}\right)\right), \tag{2.17}$$

where $\textsc{Combine}^{\ell}$ and $\textsc{Aggregate}^{\ell}$ are arbitrary differentiable combination and aggregation functions over (pairs and multisets of) numeric vectors, respectively, and $\mathcal{N}(v)$ is the set of the neighbours of node v in G. Note that for each $\mathbf{v}^{\ell} \in \mathbb{R}^{\delta^{\ell}}$, $\delta^{\ell} \in \mathbb{N}$ is the *dimension* of layer ℓ such that $\delta^0 = \delta$, and the number L of layers and the dimensions δ^{ℓ} of the hidden layers are hyper-parameters of the model. In particular, we take $\mathbf{v}^0 = \mathbf{v}$. After L iterations, the GNN outputs the annotated (undirected) graph with the same nodes and edges as G, where each node $v \in V$ is associated with a feature vector $\mathbf{v}^L$. The feature vectors of the output graph can be used to make predictions at different levels of graph components: individual nodes, edges, or an entire graph. For example, for node-level tasks such as node classification, predictions are made using $f(\mathbf{v}^L)$, where $f(\cdot)$ is a multi-class classifier such as Multi-Layer Perceptron (MLP). For edge-level tasks such as link prediction, the vectors of two nodes are often combined using an ad-hoc scoring function $g(\cdot, \cdot)$. We see KG completion as a particular case of link prediction, and will discuss the GNN-based KG completion approaches in detail in Chapter 3. One of the other examples for link prediction is the edge prediction in social networks [102]. To make predictions at graph level (e.g. graph classification), we can utilise a *pooling* function to combine the vectors of all nodes in the graph into a global representation. This pooling function can be implemented as a standard operation such as mean-pooling, max-pooling, or sum-pooling, or some more sophisticated neural networks such as Long Short-Term Memory (LSTM) Network. An illustration for the GNN message passing is provided in Figure 2.2.

The message passing framework discussed so far represents a conceptual approach shared by different GNN approaches. In what follows, we will introduce and analyse several representative GNN variants, and we first start with an early and basic GNN



(a) (b)

Figure 2.2: An example for GNN message passing. The graph in (a) contains three nodes, and each node is annotated with a representation. In the message passing step illustrated in (b), the representation of node 1 is updated by aggregating the representations from node 2 and node 3, and combining the aggregated information with its own information.

approach [46, 114]. Formally, a vanilla GNN model with $L \in \mathbb{N}$ layers and dimension $\delta \in \mathbb{N}$ updates, on each layer $\ell \in \{1, ..., L\}$, the hidden vector of node v using the rule

$$\mathbf{v}^{\ell} = \sigma^{\ell}(\mathbf{W}^{\ell}_{\text{self}}\mathbf{v}^{\ell-1} + \sum_{v' \in \mathcal{N}(v)} \mathbf{W}^{\ell}_{\text{neighbour}}(\mathbf{v}')^{\ell-1} + \mathbf{b}^{\ell}), \tag{2.18}$$

where $\mathbf{W}^{\ell}_{\text{self}} \in \mathbb{R}^{\delta^{\ell-1} \times \delta^{\ell}}$ and $\mathbf{W}^{\ell}_{\text{neighbour}} \in \mathbb{R}^{\delta^{\ell-1} \times \delta^{\ell}}$ are two learnable weight matrices, $\mathbf{b}^{\ell} \in \mathbb{R}^{\delta^{\ell}}$ is a learnable bias vector, and σ^{ℓ} is a non-linear activation function such as sigmoid, Rectified Linear Unit (ReLU), or hyperbolic tangent (tanh) function. As we can see, this basic approach simply relies on a sum operation to collect neighbourhood information, which is straightforward and easy to understand. The approach can be highly sensitive to node degrees [51], which may help capture useful structural information, but suffers from potential for instability.

Inspired by the convolution operations of CNNs, Kipf and Welling proposed Graph Convolutional Networks (GCNs) [67]. In particular, a GCN with L layers and dimension δ updates, for each layer $\ell \in \{1, ..., L\}$, the representation of node v by

$$\mathbf{v}^{\ell} = \sigma^{\ell}(\sum_{v' \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(v')|}} \mathbf{W}^{\ell}(\mathbf{v}')^{\ell-1} + \mathbf{b}^{\ell}), \tag{2.19}$$

where, similar to $\mathbf{W}_{\text{self}}$ and $\mathbf{W}_{\text{neighbour}}$ in (2.18), $\mathbf{W}^{\ell} \in \mathbb{R}^{\delta^{\ell-1} \times \delta^{\ell}}$ is also a weight matrix. Note here that the summation considers $v' \in \mathcal{N}(v) \cup \{v\}$, which adds an artificial loop

to each node in the graph and thus allows for a uniform treatment of combination and aggregation in GCNs. GCNs has become a pioneering work that contributed to the development of many modern GNNs. First, GCNs consider both topology and connectivity of the graph, and can inherently capture the structure information. Second, GCNs take a first-order approximation to the spectral convolution, which can help alleviate overfitting with fewer free parameters per filtering operation, speed up training, and provide more inductive bias. Finally, in GCNs, a renormalisation trick is employed on the graph with added self-connections, which is helpful in alleviating the numerical instabilities in the node vectors when stacking multiple GNN layers, and also have a positive impact on the training performance in practice.

Graph Isomorphism Networks (GINs) [161] is an advanced GNN architecture which studies the expressive power of GNNs and is theoretically proven to be as powerful as the Weisfeiler-Lehman graph isomorphism test. On each layer $\ell = 1, \ldots, L$, a GIN updates the vector $\mathbf{v}^\ell$ of node v using the propagation rule

$$\mathbf{v}^\ell = \mathrm{MLP}^\ell \left((1 + \epsilon^\ell)\mathbf{v}^{\ell-1} + \sum_{v' \in \mathcal{N}(v)} (\mathbf{v}')^{\ell-1} + \mathbf{b}^\ell \right), \tag{2.20}$$

where MLP^ℓ are multi-layer perceptrons, ϵ^ℓ is either a fixed numeric value as a hyperparameter, or a learnable parameter trained together with other parameters.

Attention mechanism is a popular component in numerous deep learning models, particularly in NLP [6, 82] and CV tasks [92, 137]. Graph Attention Networks (GATs) [134] apply the attention mechanism in their approach by assigning an attention score to each of the neighbours, which will be treated as a weight in the aggregation. To measure the importance of neighbouring node v' to node v, for each layer $\ell = 1, \ldots, L$, GAT first computes the coefficients for a pair of nodes using an attention function (which is, in practice, a single-layer feedforward neural network), and normalise the coefficients across all choices of v using a softmax function. Formally, the attention weights are defined as:

$$\alpha^\ell_{v,v'} = \frac{\exp(\mathrm{LeakyReLU}(\mathbf{a}^{\ell\top}[\mathbf{W}^\ell \mathbf{v}^{\ell-1} \oplus \mathbf{W}^\ell (\mathbf{v}')^{\ell-1}]))}{\sum_{v'' \in \mathcal{N}(v) \cup \{v\}} \exp(\mathrm{LeakyReLU}(\mathbf{a}^{\ell\top}[\mathbf{W}^\ell \mathbf{v}^{\ell-1} \oplus \mathbf{W}^\ell (\mathbf{v}'')^{\ell-1}]))}, \tag{2.21}$$

where $\mathbf{a}^\ell$ is a learnable vector, $\mathbf{W}^\ell$ is a learnable matrix, $\cdot^\top$ is the transposition operation, and $\oplus$ is the concatenation operation. Finally, we define how GAT updates the feature vector of a node. GAT employs a multi-head attention, and more specifically, K independent attention functions a^ℓ to compute K different coefficients,

and concatenate together all the resulting feature vectors. Formally, for each layer $\ell = 1, \ldots, L-1$, the GAT updates the vector $\mathbf{v}^\ell$ of node v using the propagation rule

$$\mathbf{v}^\ell = \bigoplus_{k=1}^{K} \sigma^\ell \left(\sum_{v' \in \mathcal{N}(v)} (\alpha_{vv'}^\ell)^k \mathbf{W}_k^\ell (\mathbf{v}')^{\ell-1} + \mathbf{b}^\ell \right), \tag{2.22}$$

where $\bigoplus$ is a concatenation function and $\mathbf{W}_k^\ell \in \mathbb{R}^{K\delta^{\ell-1} \times \delta^\ell}$ is the corresponding weight matrix learnt for head k. For the final layer $\ell = L$, instead of concatenation, GAT updates the vector of node v by averaging the K resulting feature vectors:

$$\mathbf{v}^L = \sigma^L \left(\frac{1}{K} \sum_{k=1}^{K} \sum_{v' \in \mathcal{N}(v)} (\alpha_{vv'}^L)^k \mathbf{W}_k^L (\mathbf{v}')^{L-1} + \mathbf{b}^L \right). \tag{2.23}$$

The attention mechanism adopted in GATs enables the model to assign different importance scores to different neighbouring nodes, allowing the network to concentrate on the most relevant information within the graph. Furthermore, these attention scores provide valuable insights, enhancing our comprehension of the neural network's decision-making process and thus improving the model's explainability. Nevertheless, one of the key bottlenecks of GATs is that they can be computationally intensive, in particular when dealing with large graphs or using a large number of attention heads. Moreover, another limitation of GATs is that they can not capture long-range dependencies. To address such limitation, Transformer [133], which is a very popular deep learning architecture in recent years and serves as a de-facto backbone of several state-of-the-art NLP systems such as BERT [37] and GPT-3 [20], has also been exploited in many works improving GNNs, since it allows models to attend to any part of the input sequences. Concrete examples include Graphormer [169] and UniMP [119]. We refer the readers to Sun et al. [123] for a recent review of attention-based and transformer based GNNs.

GNNs have a variety of advantages compared with conventional neural networks. First of all, the message passing framework aggregates and propagates information across connections between nodes, and can thus effectively capture structure information such as node degrees and dependencies in a graph. In addition, GNNs have built-in inductive capabilities for graph-structured data: they are invariant under isomorphisms and insensitive to identity of nodes. Hence, they can generalise well to unseen nodes and graphs, making them appropriate for applications with varying graph structures. In Chapter 3 and 4, we will use GNN as an important component of our approach for inductive KG completion and inductive knowledge-enhanced recommendation. Finally, GNNs are proven to be better extrapolators which achieved

good performance on reasoning tasks [162]. In Chapter 5, we will also explore the capability of GNN-based models to capture inference patterns. These advantages make GNNs a powerful tool for a wide range of applications, including applications in scenarios using data with explicit relational structure, such as KG completion [97, 115] and recommender systems [131, 177]; and in scenarios using data where relational structure is implicit or absent, such as semantic segmentation for images [105, 156], and sentiment analysis for text [76, 173].

Despite the evident strengths of GNNs, they also have certain inherent limitations that require careful considerations. First of all, the scalability of GNNs remains a substantial challenge as they struggle to efficiently handle large-scale graphs (e.g., with billions of nodes) in practice, limiting their utility in tasks involving deep networks with numerous parameters and huge datasets. In addition, deep GNNs may suffer from the *over-smoothing* problem when stacking multiple layers [27, 31], causing nodes to converge to similar or indistinguishable feature vectors in the deeper layers, and thus compromising their ability to effectively capture graph structure information and produce meaningful representations. Moreover, like other deep learning approaches, GNNs are susceptible to the *overfitting* problem when the dataset is small or noisy, which requires application of appropriate regularisation tricks and careful training strategies. Last but not least, as the message passing paradigm relies heavily on the graph edges to propagate information, GNNs may struggle with nodes that have few or no connections, and we refer to this as the *cold-start* problem. All in all, how to improve the message-passing paradigm of GNNs in a deeper, more expressive, and more scalable fashion, remains a substantial challenge and a compelling subject for exploration in this field.

2.6 Mathematical Notations

In Table 2.1, we provide a list of commonly used notations throughout this thesis. Additional notations will be introduced in each of the following chapters if necessary.

Table 2.1: Mathematical notations.

Notation	Explanation
$\mathcal{C}$	Set of conclusion triples
$\mathcal{I}$	Set of items
$\mathcal{K}$	A knowledge graph (KG)
$\mathcal{M}_{\mathcal{R}}(\mathcal{K})$	Materialisation of program $\mathcal{R}$ on KG $\mathcal{K}$
$\mathcal{N}$	Set of negative triples
$\mathcal{N}(v)$	Set of neighbourhoods of node v
$\mathcal{P}$	Set of positive triples
$\mathcal{R}$	A program or rule set
$\mathcal{T}_r(\mathcal{K})$	One-step rule application for the rule r on KG $\mathcal{K}$
$\mathcal{U}$	Set of users
E	Edge set of the graph
G	A graph
$G_{\mathcal{K}}^{\Lambda}$	Annotated graph generated for KG $\mathcal{K}$ and candidates set Λ
N	Set of negative examples
Q	Set of edges representing interactions between users and items
V	Node set of the graph
$\mathbf{W}$	A weight matrix
$\mathbf{b}^{\ell}$	A bias vector specified to layer ℓ
$\mathbf{v}^{\ell}$	A vector of node v specified to layer ℓ
$(\mathbf{v}^{\ell})_i$	The i-th element of vector $\mathbf{v}^{\ell}$
δ	Annotation dimension
Λ	The set of candidate triples
σ	An assignment
ϱ	A renaming
$\preceq$	The lexicographic order
$\mathsf{Consts}(\mathcal{K})$	Constants used in KG $\mathcal{K}$
$\mathsf{Preds}(\mathcal{K})$	Predicates used in KG $\mathcal{K}$
$\mathsf{Rels}(\mathcal{K})$	The set of relations of KG $\mathcal{K}$
$\mathsf{Sig}(\mathcal{K})$	The signature of KG $\mathcal{K}$
$\mathsf{Types}(\mathcal{K})$	The set of types of KG $\mathcal{K}$
$\_\mathsf{R}, \_\mathsf{S}, \_\mathsf{T}$	Relation templates
$\_\mathsf{t}$	Type template
$\mathfrak{p}$	An inference pattern

Chapter 3

INDIGO: GNN-Based Inductive Knowledge Graph Completion Using Pair-Wise Encoding

3.1 Introduction

In response to the ever-changing nature of information in the real world, KGs are evolving over time, and they may be extended with triples describing new objects [32, 62]. For instance, biomedicine KGs are often used to describe the relations between objects such as genes, proteins, diseases, and drugs [74]. These KGs are continuously updated to incorporate new research findings and the discovery of new drugs. This evolution nature has sparked interest in research about KG-oriented tasks in the *inductive* setting, where the KGs can be extended with triples involving new constants after training. Particularly, in this setting, the approaches are usually required to be applicable to triples involving new constants without retraining. This requirement is especially relevant for production-ready machine learning models [127], where frequent retraining can significantly increase the cost of the products using these models. Most approaches to KG completion, such as TransE [18], DistMult [164], and RotatE [124], are based on graph embedding techniques. These approaches are transductive, and they are not applicable to the inductive setting since the parameters learnt by models are tied to the constants in the original KG. For them to become applicable, a retraining would be required using also the newly introduced triples. In order to tackle this challenge, in this chapter, we consider the task of inductive KG completion, which was introduced in Section 2.1.

There are a variety of methods for inductive KG completion, and each type of method suffers from its own shortcomings. For example, rule-based methods [89, 111,

166] show good results, but only when the shape of the rules is known in advance. Some methods utilise side information about unseen constants, such as attributes [52] and textual descriptions [34, 118, 136, 157, 168], but such information is not always available in practice. Recent approaches, such as R-GCN [115], the approaches in Hamaguchi et al. [50] and Wang et al. [145], GraIL [127], and NBFNet [181], aim to overcome these limitations by exploiting GNNs. Nevertheless, existing GNN-based approaches still possess certain limitations: most of them require predefined scoring functions to make predictions based on the output of the GNNs [50, 115, 127, 145]; the predictions of a triple often depend on the standalone neighbourhoods of its constants in the input KG [50, 115, 145]; and sometimes they suffer from the scalability issue [127].

In this chapter, we propose a novel inductive KG completion approach where the KG is encoded into the input graph to a GNN (for which in practice we use a GCN [67]) in a transparent and direct way, so that the inductive capabilities of the GNN are fully exploited. In contrast to existing approaches, our encoding establishes a one-to-one correspondence between elements of the feature vectors in the innermost and outermost layers of the GNN and triples over the KG's signature, and hence the predicted triples can be read out directly from the outermost layer without the need for an external scoring function. As a result, our approach has neither the aforementioned drawbacks of scoring-based approaches, nor the bottleneck of GraIL, being able to process the entire graph at once; moreover, our approach allows to make predictions for several triples in one run, which may provide significant speed up in practice. We have implemented our approach in a system called *INDIGO* (*Inductive Knowledge Graph Competion*), and compared it with R-GCN, GraIL, and the system proposed by Hamaguchi et al. [50] on the inductive benchmarks developed in Teru et al. [127] and Hamaguchi et al. [50], as well as on a new benchmark we have developed based on Freebase. Our results show that INDIGO not only outperforms the baselines on these benchmarks, but can also be trained and applied more efficiently. We also conducted an ablation study to investigate the impact of different GNN variants on the system performance. We have finally studied the ability of our system to learn in practice common inference patterns represented using logical rules; our experiments show that our system is able to learn more comprehensive rule sets than the evaluated baselines.

The rest of the chapter is organised as follows. As the KG completion problem is already defined in Section 2.1, we will begin with a review on related work about KG completion in Section 3.2, and then describe the details of our approach to inductive

KG completion in Section 3.3. In Section 3.4, we highlight the key insights behind our main contribution, namely, the pair-wise encoding, with a dicussion about GNN-based KG completion methods using unary and binary encoders. Then, in Section 3.5, we will show a comprehensive evaluation of our INDIGO system developed based on the approach. We will conclude this chapter by a summary of our work in Section 3.6.

3.2 Related Work

Recently, KG completion has garnered considerable attention within the academic community, for its vital role in enhancing data quality and facilitating advanced analysis of KGs. From Section 3.2.1 to 3.2.4, we will review the literature on KG completion, which broadly falls into three categories: embedding-based, rule-based, and GNN-based KG completion. It is also worth mentioning that in this work, we assume that only the structural information of the KG (i.e., triples) are provided, and KG completion approaches we compare should rely only on such structural information. However, there is also some literature exploiting side information for KG completion, such as attribute information or text description, and we will provide a brief view of related techniques in Section 3.2.5. Note that in these approaches, the embedding-based approaches are only transductive, while the rule-based and GNN-based approaches, and the approaches using side information are also applicable to the inductive setting.

3.2.1 Embedding-Based KG Completion

Generally speaking, embedding-based KG completion approaches rely on the graph embedding techniques, which first embed the KG into a low-dimensional vector space and then generate the predicted triples by applying a predefined scoring function to the learnt vectors. It is worth noting that, the embedding-based KG completion systems exhibited remarkable performance in transductive settings, but are not applicable to the inductive settings, since the learnt parameters are tied to the constants in the original KG. There are three representative types of embedding-based KG completion: translational methods, tensor factorisation methods, and neural methods.

3.2.1.1 Translational Methods

In translational methods, relations are modelled as translations between the vectors of subjects and objects in the low-dimensional vector space. They usually define a scoring function $f(\cdot, \cdot, \cdot)$ to measure the distance between the vector of the subject

combining the vector of the relation, and the vector of the object. Then, they rely on the scoring function to determine the probability that a KG triple holds.

TransE [18] is a pioneer work in this field and have had a far-reaching and enduring impact on the subsequent translational approaches. Given a triple (s, R, o), TranE first maps it into a δ-dimensional vector space. More specifically, it models the subject s, relation R, and object o with vectors $\mathbf{s} \in \mathbb{R}^\delta$, $\mathbf{R} \in \mathbb{R}^\delta$, and $\mathbf{o} \in \mathbb{R}^\delta$, respectively. Then, TransE compute the score of the triple as

$$f(\mathbf{s}, \mathbf{R}, \mathbf{o}) = ||\mathbf{s} + \mathbf{R} - \mathbf{o}||. \tag{3.1}$$

While TransE has been effective in many KG completion benchmarks, there are still some inherent limitations. First, the limited expressiveness of TransE makes it difficult to deal with *one-to-many*, *many-to-one*, and *many-to-many* relations. Second, TransE is primarily designed for the situation where there is only one relation between two constants, and two relations will be mapped to identical vectors when they both hold for the same pair of constants. Finally, TransE may struggle to model inverse relations accurately. In response to these limitations, there has been a continuous effort in the field to advance translational methods, such as TransH [149], TransR [77], and TransD [63].

Inspired by Euler's identity, RotatE [124] employs complex-value vectors instead of real-value vectors, and represents relations as rotations between constants in the complex vector space. In particular, for a triple (s, R, o), RotatE first embed the subject s, relation r and object o into complex vectors $\mathbf{s} \in \mathbb{C}^\delta$, $\mathbf{R} \in \mathbb{C}^\delta$, and $\mathbf{o} \in \mathbb{C}^\delta$, respectively. The score of the triple as

$$f(\mathbf{s}, \mathbf{R}, \mathbf{o}) = ||\mathbf{s} \circ \mathbf{R} - \mathbf{o}||, \tag{3.2}$$

where $\circ$ denotes the Hadamard (element-wise) product. Thanks to its complex embedding techniques and rotational nature, RotatE becomes more expressive than TransE and is shown theoretically to model inference patterns such as symmetry, inversion, and composition, which are introduced in Section 2.4.

BoxE [1], a spatio-translational embedding approach, is proposed to further improve the expressivity of translational methods and support more inference patterns such as hierarchies. In BoxE, the relations are represented as boxes or hyper-rectangles; and each entity is represented as a point in these boxes or hyper-rectangles. A triple (s, R, o) is true if the points for s and o appear in their respective R boxes. BoxE is theoretically proven to capture symmetry, anti-symmetry, hierarchy, inversion, and intersection. However, BoxE is not able to capture composition in theory [1].

3.2.1.2 Tensor Factorisation Methods

Tensor factorisation methods are a class of methods that utilise tensors to model the relationships between constants in the KG. In particular, a KG can be represented as a three-dimensional binary tensor $\mathbf{Y} \in \{0,1\}^{|\mathsf{Consts}(\mathcal{K})|\times|\mathsf{Rels}(\mathcal{K})|\times|\mathsf{Consts}(\mathcal{K})|}$. We fix an (arbitrary) *enumeration* id_c assigning a unique natural number from 1 to $|\mathsf{Consts}(\mathcal{K})|$ to each $c \in \mathsf{Consts}(\mathcal{K})$, and analogously, an enumeration id_r from 1 to to $|\mathsf{Rels}(\mathcal{K})|$ to each $R \in \mathsf{Rels}(\mathcal{K})$. The tensor is populated with information about the existence of a triple. Formally, if a triple (s, r, o) holds, we have $\mathbf{Y}_{\mathsf{id}_c(s),\mathsf{id}_r(R),\mathsf{id}_c(o)} = 1$. Tensor factorisation methods seek to learn the low-dimensional representations of constants and relations, so that they can generate a tensor $\mathbf{X}$ by combining the representations of constants and relations using some scoring functions (e.g. dot product, bilinear interactions) to approximate the ground-truth tensor $\mathbf{Y}$.

RESCAL [100] is one of the early embedding tensor factorisation approaches. In RESCAL, for a triple (s, R, o), it models the relation R with a asymmetric matrix $\mathbf{M}_R \in \mathbb{R}^{\delta\times\delta}$, and the constants s and o with vectors $\mathbf{s} \in \mathbb{R}^\delta$ and $\mathbf{o} \in \mathbb{R}^\delta$, respectively. Then, the score of the triple is computed as

$$f(\mathbf{s}, \mathbf{R}, \mathbf{o}) = \mathbf{s}^\intercal \mathbf{M}_R \mathbf{o}, \tag{3.3}$$

where $^\intercal$ denotes the matrix transposition operation. Based on the structure of RESCAL, DistMult [164] restricts $\mathbf{M}_R$ to be a diagonal matrix to reduce the number of relation parameters. This restriction makes DistMult more computationally efficient and easier to train.

ComplEx [129] further makes use of complex valued embeddings to improve the expressiveness of the tensor factorisation. Particularly, in ComplEx, the relation matrix $\mathbf{M}_R$ and constant vectors $\mathbf{s}$ and $\mathbf{o}$ are defined in the complex space, so that $\mathbf{M}_R \in \mathbb{C}^{\delta\times\delta}$ and $\mathbf{s}, \mathbf{o} \in \mathbb{C}^\delta$. It then computes the score of the triple (s, R, o) using

$$f(\mathbf{s}, \mathbf{R}, \mathbf{o}) = Re(\mathbf{s}^\intercal \mathbf{M}_R \bar{\mathbf{o}}), \tag{3.4}$$

where $Re(\cdot)$ denotes the real part of a complex value, and $\bar{\cdot}$ denotes the conjugate. While maintaining the computational efficiency and scalability of DistMult, ComplEx is able to capture more complex relationships such as symmetric or antisymmetric relations. Such explorations are also extended to more complex vector space such as quaternion [179] and even dual quaternion space [25], and they have shown state-of-the-art performance on several KG completion benchmarks.

3.2.1.3 Neural Methods

The advances in neural network architectures have led to significant improvements in many AI tasks. In general, neural methods incorporate the neural network structure to the process of score computing, and have emerged as powerful solutions for KG completion. Neural Tensor Networks (NTN) [120], is one of the pioneering neural methods. In NTN, a bilinear tensor layer is employed to establish a direct connection between the two constant vectors across various dimensions. For a triple (s, R, o), NTN computes its score by

$$f(\mathbf{s}, \mathbf{R}, \mathbf{o}) = \mathbf{u}^\intercal \tanh(\mathbf{s}^\intercal \mathbf{W}_R^{[1:k]} \mathbf{o} + \mathbf{V}_R \begin{bmatrix} \mathbf{s} \\ \mathbf{o} \end{bmatrix} + \mathbf{b}_R), \tag{3.5}$$

where $\mathbf{W}_R^{[1:k]} \in \mathbb{R}^{\delta \times \delta \times k}$, $\mathbf{V}_R \in \mathbb{R}^{k \times 2\delta}$ are learnable parameters and $\mathbf{b}_R \in \mathbb{R}^k$ is a bias vector. NTN leverages a relatively simple neural network architecture, and there is a number of follow-up work exploiting more advanced neural network architectures. For instance, ConvE [36] utilises a multi-layer two-dimensional convolutional network to model the interactions between the constants and relations in the KG, and ConvKB [99] further extends ConvE by the relations among same dimensional entries of the embeddings. Although neural methods have seen significant success in this field, they also pose some challenges due to the neural nature. First of all, neural methods often require a large amount of training data compared with other approaches, and may not perform well when the KGs are sparse or noisy. Hence, they are more suitable for the situations where there are sufficient amounts of data. Moreover, neural methods can be susceptible to overfitting, and in particular for more sophisticated structures with a large number of parameters. Thus, appropriate regularisation and tuning are needed to mitigate this issue. Finally, neural methods are *black-box* in the sense that the internal details of the model lack transparency, thus making it difficult to interpret why the predictions are made. A potential solution may be to incorporate neural methods with symbolic rules.

3.2.2 Discussion

So far, we have reviewed the general idea and prominent examples of embedding-based KG completion approaches. As mentioned in Section 3.1, one key limitation of these approaches is that they are not applicable to the inductive settings. For them to become applicable, it would require retraining using also the newly introduced triples. Moreover, as we can see, these embedding-based systems seek to provide a semantically rich representation of constants and relations in the KG. There are also

some literature [1, 18, 26] discussing in theory about the systems' ability to capture inference patterns, such as symmetry, hierarchy, and composition. However, these theoretical results offer limited insight into their capability to capture the patterns in practice. Therefore, in Chapter 5, we will propose new benchmarks that take into account the inference patterns, and conduct comprehensive evaluation on a series of KG completion systems.

3.2.3 GNN-Based KG Completion

GNNs have demonstrated significant success across a wide range of tasks with graph structured data, and there is also an extensive literature on the use of GNNs for KG completion. In general, we see GNN-based KG completion approaches as working in three stages. First, they *encode* a KG into a graph where each node is annotated with a low-dimensional feature vector. Then, the encoded graph is fed into a GNN. Finally, the predicted triples are *decoded* from the GNN's output. The key differences among these approaches lie in their way of encoding, the GNN architecture, and their tailored scoring function for decoding. In this section, we will first review several prominent GNN-based KG completion approaches, and then conclude with a thorough discussion about the advantages and disadvantages of these approaches.

R-GCN [115] is one of the earliest GNN-based KG completion approaches. To address the limitation of vanilla GCNs that they are not able to handle multi-relational data, they propose to aggregate information in a relation-specific manner. In particular, R-GCN first encodes each constant of the KG into a node with a randomly initialised feature vector, and each triple into a directed coloured edge between the nodes. Then, the GNN aggregate information separately for each of the colours. Formally, for each layer $\ell = 1, \ldots, L-1$, R-GCN updates the vector $\mathbf{v}^\ell$ of node v using the propagation rule

$$\mathbf{v}^\ell = \sigma^\ell \left(\sum_{c \in \mathsf{Col}} \sum_{v' \in \mathcal{N}_c(v)} \frac{1}{\epsilon_c} \mathbf{W}_c^\ell (\mathbf{v}')^{\ell-1} + \mathbf{W}_{\text{self}}^\ell \mathbf{v}^{\ell-1} \right), \tag{3.6}$$

where Col is a set of colours, $\mathcal{N}_c$ is the set of nodes connected to v by an edge in colour c, and ϵ_c is constant that can either be learned or chosen in advance. For decoding, they use DistMult [164] as the scoring function, where each relation is associated with a diagonal matrix $\mathbf{M}_R$, and the score of a triple (v_1, R, v_2) is computed as

$$f(\mathbf{v}_1, \mathbf{R}, \mathbf{v}_2) = \mathbf{v}_1^\intercal \mathbf{M}_R \mathbf{v}_2. \tag{3.7}$$

Similar to R-GCN, SACN [116] is also based on the idea of assigning different weights to different relations. It encodes a KG into multiple subgraphs, where the edges within each subgraph represent triples associated with a particular relation. Then, it aggregates information for each of the subgraphs, and the embeddings from different subgraphs are summed up using a relation weight α, which is also a training parameter. SACN finally develops a Conv-TransE scoring function as the decoder.

While both approaches have demonstrated good performance in some KG completion benchmarks, they also have several limitations that need to be taken into account.

First of all, in both of the approaches, each neighbouring node (connected by the same coloured edges) contributes equally when aggregating information. To address such limitation, KBGAT [97] follows a similar way of encoding to R-GCN, but incorporates an attention mechanism that allows it to assign different attention weights to different neighbouring nodes.

Furthermore, in essence, the way of encoding for these two approaches only allows them to update the embedding for each constant in the KG using the GNN. Furthermore, the operation of representing relations as matrices may result in over-parameterisation. The shortcomings are overcome by CompGCN [132], where the relations (colours) are also involved in the GNN aggregation. Formally, for each layer $\ell = 1, \ldots, L-1$, CompGCN updates the vector $\mathbf{v}^\ell$ of node v and vector $\mathbf{R}_c^\ell$ of colour c using the rule

$$\mathbf{v}^\ell = \sigma^\ell \left(\sum_{c \in \mathsf{Col}} \sum_{v' \in \mathcal{N}_c(v)} \mathbf{W}_c^\ell \phi \left((\mathbf{v}')^{\ell-1}, \mathbf{R}_c \right) \right), \tag{3.8}$$

where $\phi(\cdot, \cdot)$ is a composition operation. With this design, compared with R-GCN and SACN, CompGCN is more parameter efficient and can thus effectively mitigate the overfitting issue.

Finally, their prediction for a triple relies on the standalone neighbourhoods of its constants in the input KG, without considering the shared elements of these neighbourhoods. In general, the approaches such as R-GCN, SACN, and CompGCN use a *unary* encoder to learn individual representations of the constants, and then rely on a *binary* decoder to make predictions based on the learnt representations. However, such encoder-decoder architecture can not capture either neighbourhood-overlap based features, or the distance information. Moreover, from the theoretical point of view, it is proved that R-GCN and CompGCN is upper bounded by multi-relational 1-dimensional Weisfeiler-Leman algorithm (1-RWL) for the capacity of R-GCN and CompGCN to distinguish nodes in the KGs; and over every KGs, there is a R-GCN

or CompGCN model that is as powerful as 1-RWL. Nevertheless, there still exist multi-relational graphs that neither R-GCN nor CompGCN can distinguish [9]. We refer the readers to Zhang et al. [178], Barcelo et al. [9], and Huang et al. [60] for a thorough theoretical analysis about this issue, and we will further discuss it in detail in Section 3.4.

The limitation about unary encoding is resolved by a KG completion system called GraIL [127], which is rooted from SEAL [176], a link prediction approach. To predict if a triple holds in a KG, GraIL first extracts a subgraph of the KG for that triple, and encodes the subgraph in a way similar to R-GCN, except that the vectors of the nodes are initialised based on their distance to the nodes that correspond to the constants in the triple. Then, it uses a GNN to convey information and update the feature vectors, and makes prediction for a triple using a scoring function applied globally to the output vectors of all nodes in its dedicated subgraph. There are also recent endeavours making further improvements based on the idea of GraIL. For instance, CoMPILE [86] extracts a directed subgraph instead of an undirected one, and devises an entity-relation communicative message passing network to replace the original GNNs. In addition, TACT [28] further considers the correlation between relations based on the topological structures to strengthen the encoding of the subgraph. A common bottleneck of the above three methods is that they require generating a subgraph for each triple they would like to predict if it should be added to a KG, which becomes very time-consuming when there are a large number of triples. Some recent path-based approaches using GNNs also achieved promising results in the inductive KG completion task, such as PathCon [138] and NBFNet [181]. PathCon relies on a relational context module to aggregate the neighbourhood information for the target pair, and a relational path module to characterise the relative position between the target pair in the KG. Inspired from the generalised Bellman-Ford algorithm [8, 12] for finding the shortest paths, NBFNet learns the pairwise node representations by summarising the path representations between the node pair. An analysis about the expressive power of NBFNet is provided by Huang et al. [60] by aligning it with the relational asymmetric local 2-WL (rawl_2). In general, the methods with subgraph extraction such as GraIL, and the methods based on paths such as NBFNet, can both be viewed as frameworks using a *binary* encoder learning pairwise representations conditioned on the candidate triple, and a unary decoder to make predictions based on the representations [60]. We will also discuss it with examples in detail in Section 3.4.

Compared with embedding-based approaches, GNN-based approaches exhibit the inherent inductive capability in favour of generalisability due to the nature of their

design. In particular, GNNs are biased towards graph symmetries (nodes with same neighbourhoods receive the same value). Moreover, they are invariant under isomorphisms (i.e., insensitive to the identity of nodes) and can capture general patterns represented as logic formulas [9, 10, 60, 94, 161], so that the representations are not dependent on the specific node or edge order within a graph. Such inductive capability allows GNN-based approaches applicable to evolving KGs, which is very common in real-world practice, such as KGs with newly published books or newly launched organisations. Another

However, there are still some open challenges for GNN-based approaches. On the one hand, most of them still rely on a tailored scoring function to make predictions from the output of the GNN. On the other hand, as discussed in the example of GraIL, they often suffer from scalability issues and can be very computationally expensive. It is also remarkable that the literature has seen relatively rare exploration on these GNN-based approaches' ability to capture inference patterns, leaving a notable research gap that beckons for in-depth investigation.

3.2.4 Rule-Based KG Completion

Rule-based KG completion approaches aim to first extract a set of probabilistic rules from a KG, and then infer missing triples based on the extracted rules. These approaches typically rely on a set of predefined syntactic shapes, and mine rules of such specific shapes from the KG.

RuleN [89] is a simple rule-based system that supports the following two types of rules:

$$(x_1, R_1, x_2) \wedge \cdots \wedge (x_n, R_n, x_{n+1}) \rightarrow (x_1, S, x_{n+1}), \tag{3.9}$$

$$(x, R, y) \rightarrow (x, R, a), \tag{3.10}$$

where x_i and y are variables, a is a constant, and R_i and S are relations. It uses a twofold sampling approach to search rules satisfying the shapes and computes the confidence of the rules. To predict the missing entry of the triple $(c, R, ?)$, RuleN selects the rules with relation R in the head, looks up all body groundings in the KG by constituting the first variable in the atom of rule head with c, and computes the ranking of all possible candidates based on the confidences. AnyBURL [88] is developed based on RuleN but has an *anytime behaviour*, which is able to return valid results even if it is interrupted before it ends, and the the quality of the results improves over time. It is worth mentioning that there is also some work learning rules in an end-to-end fashion. Such examples include Neural-LP [166] and DRUM [111].

Overall, a significant advantage of these rule-based approaches is that they are interpretable since the extracted rules are human-readable and can thus serve as possible explanations for the predictions. Furthermore, the rule-based approaches are inherently *inductive*: in most cases, the learnt rules capture constant-independent relation semantics, and can thus be applied to arbitrary constants. However, one limitation of rule-based approaches is that they often suffer from the scalability issue. Moreover, the prediction performance relies heavily on the quality and completeness of the syntactic shapes that are required to know in advance, which are usually very limited.

3.2.5 KG Completion Using Side Information

The aforementioned types of approaches all assume that only the structure information (KG triples) can be leveraged to complete the KG. However, there is also some literature exploiting side information for KG completion. Concrete examples of such side information include constant attributes and text description (or lexical information). Please note that in some definitions, the information about attributes or text description is also represented as triples.

DEAL [52] exhibits a successful practice of using attribute information, which deploys an attribute-oriented encoder to leverage the entity attributes, and a structure-oriented encoder (such as GNN) to capture information from the graph structure, and finally uses an alignment mechanism to combine information from both encoders.

The approaches utilising text description of the constants mainly differ in the way of encoding text information. For example, DKRL [157] utilises a continuous bag-of-words encoder (CBOW) and a CNN to encode the semantics of text description. ConMask [118] proposes a relation-dependent content masking mechanism to select small relevant snippets and filter out irrelevant text based on the relation. Chen et al. also uses Word2Vec-like language models to utilise the lexical information for learning the KG embeddings [29, 30].

Inspired from the success of Large (pre-trained) Language Models (LLMs) [20, 37, 108, 109], recent studies have also harnessed LLMs as a key point of explorations for KG related tasks KG embeddings, KG completion. For example, KG-BERT [168] is one of the early works in this field, which is based on the bidirectional encoder representations from transformers (BERT) [37], and uses the name or text description of the constants as the input sentence of the pre-trained model for fine-tuning. StAR [136] and BLP [34] further combines such paradigm with the graph embedding techniques, so that the system can benefit from both text information and structure

information. Several popular machine learning strategies are also incorporated to further improve the performance, such as contrastive learning [144] and multitask learning [66], and they have achieved competitive performance on a series of benchmarks.

The approaches utilising side information are inductive in nature, since the predictions can be made based on the provided attribute information or text description about the unseen constants. They are helpful in particular when the unseen constants are isolated in the incomplete KG during the testing stage, whereas the other inductive KG completion systems using only structural information are not able to deal with this case. The limitation of these types of approaches is also evident: they only work well when such side information is available. Furthermore, fine-tuning LLMs require substantial computational resources, which is a potential bottleneck for the development of future research.

3.2.6 Summary

In this section, we have reviewed the literature about embedding-based, rule-based, GNN-based KG completion approaches, and approaches using side information. In addition to introducing the technical details, we have also analysed the advantages and limitations for each of the approaches. In summary, it is still an open challenge to develop an inductive KG completion system without either the scoring function design or the scalability issue. In the rest of this chapter, we will propose a novel GNN-based approach to address the aforementioned limitations. Moreover, it is worth noting that the current research landscape lacks comprehensive exploration of these approaches' ability to capture inference patterns in practice. Hence, in Chapter 5, we will propose a new benchmarking approach that enables researchers to create inferential benchmarks and assess the performance of their own KG completion systems.

3.3 Inductive KG Completion Using Pair-Wise Encoding

In this section, we introduce our approach to inductive KG completion, where the completion function $f(\cdot, \cdot)$ is realised by the following three components.

- A pair-wise encoder, which takes an (incomplete) KG $\mathcal{K}$ and a set Λ of candidate triples of the same signature as input and returns a node-annotated graph $G^{\Lambda}_{\mathcal{K}}$.

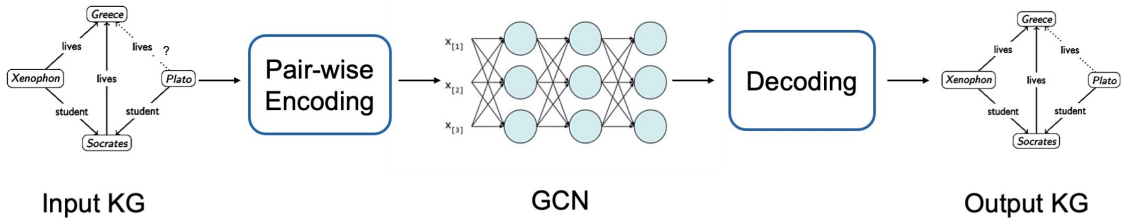


Figure 3.1: An illustration for the pipeline of our approach

The encoder is pair-wise in the sense that each node in the annotated graph corresponds to a pair of constants occurring in $\mathcal{K}$ and Λ.

- A GNN model, which updates the annotated feature vectors in graph $G^{\Lambda}_{\mathcal{K}}$ by aggregating the annotations of the neighbouring nodes; in our approach, we use a GCN variant [67] due to its effectiveness and efficiency.

- A decoder, which extracts the predictions $f(\mathcal{K}, \lambda)$ for each $\lambda \in \Lambda$ from the updated graph output by the GNN; this step is essentially mirroring the encoding.

The details of these components are given in Sections 3.3.1, 3.3.2, and 3.3.3, respectively. Figure 3.1 depicts an overall pipeline of our approach.

The key conceptual difference distinguishing our approach from existing inductive systems lies in the encoding and decoding steps. First, our approach encodes a *pair* of constants to a single node in the graph; this is in contrast to other approaches such as R-GCN and GraIL, where each constant is directly encoded to a single node in the graph. It is inherently more consistent with the nature of knowledge graph completion, which tries to predict if the link holds between a pair of constants. Second, in previous approaches, the node annotations are typically initialised randomly, with one-hot encodings of the constants' index, or based on relative distance. However, in our approach, each element of the node annotations is initialised based on the existence of a KG fact involving the corresponding constant pairs. In this way, we can explicitly encode the relevant structural information to the node annotations. Furthermore, existing approaches typically rely on ad-hoc scoring functions to decode the output of the GNN, whereas in our approach the predicted triples can be read

out *directly* from the feature vectors in the final layer of the GNN. Finally, in our approach, we are able to make predictions for a set Λ of candidate triples at once (for the same KG) rather than for a single triple; this facilitates training and testing in many practical scenarios, including those reflected in existing benchmarks.

Note that the node-annotated graphs, which are defined in Section 2.5, are used on all three components of our approach. There are significant differences between KGs, which represent graph-structured data, and annotated graphs, which are the graphs manipulated by GNNs. Nodes in a KG represent constants and types, and edges are labelled with relations between such entities; in contrast, nodes in annotated graphs are annotated with feature vectors, and (undirected and unlabelled) edges between nodes indicate that the values of their vectors influence each other during the execution of the GNN.

In our approach, the annotation dimension δ is determined by the relevant sets Rels of relations and Types of types (which are assumed known and fixed before training in the inductive setting); in particular, we take $\delta = |\mathsf{Types}| + 2 \cdot |\mathsf{Rels}|$. This allows us to associate positions in feature vectors to types, relations, and inverses of relations, which will be instrumental for encoding and decoding. In particular, we fix an (arbitrary) *enumeration* id assigning a unique natural number from 1 to δ to each $t \in \mathsf{Types}$, each $r \in \mathsf{Rels}$, and the *inverse* r^- of each $r \in \mathsf{Rels}$, and assume that, for every type, relation, or inverse p, element $\mathsf{id}(p)$ of a node feature vector corresponds to p. Observe that the dimension δ does not depend on the constants of the input KG; so our approach is indeed inductive: once trained, our model is applicable to any KG over the fixed sets of types and relations, but using arbitrary constants. For the rest of this section, we assume that Rels, Types, id, and δ are fixed. (We can also say that the *ontology* is fixed if only terminological knowledge is used.)

3.3.1 Pair-Wise Encoder

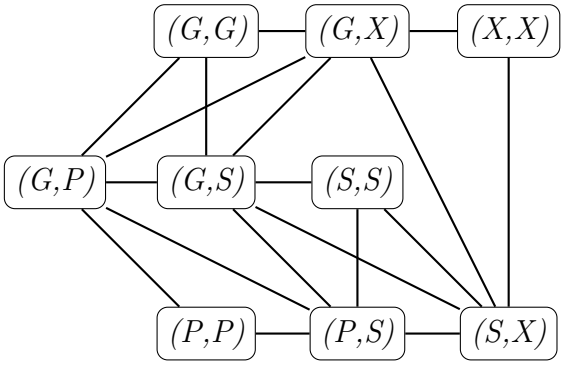
Our encoder maps a KG $\mathcal{K}$ and a set Λ of candidate triples to a δ-annotated graph $G^{\Lambda}_{\mathcal{K}}$ where each node $u_{c,d}$ corresponds to a pair of (not necessarily distinct) constants c, d connected by a triple in $\mathcal{K}$ or Λ. To avoid duplication of information, graph $G^{\Lambda}_{\mathcal{K}}$ contains only one of the nodes $u_{c,d}$ and $u_{d,c}$ when $c \neq d$. The choice between $u_{c,d}$ and $u_{d,c}$ is immaterial to our approach; for definiteness, we employ the lexicographic order $\preceq$ on the set of constants and add node $u_{c,d}$ to $G^{\Lambda}_{\mathcal{K}}$ only if $c \preceq d$. As formalised in the following definition, the annotation of $u_{c,d}$ is a Boolean feature vector where an element is set to 1 if and only if the corresponding type, relation, or inverse relation holds in $\mathcal{K}$ for c and d (e.g., $(\mathbf{u}_{c,d})_i = 1$ if and only if $c = d$ and $(c, \mathsf{type}, t) \in \mathcal{K}$ for type

t with $\mathsf{id}(t) = i$); if c and d appear in the same triple only in Λ, then the elements of $\mathbf{u}_{c,d}$ are all 0. As a result, our encoding establishes a one-to-one correspondence between elements of the feature vectors and relevant triples over the KG's signature. Finally, two nodes in $G^{\Lambda}_{\mathcal{K}}$ are connected by an edge if they share a constant, which, as we will see, allows GNNs to learn structural patterns in the data (e.g., such as the pattern in Figure 2.1 capturing that teachers live in the same country as their students).

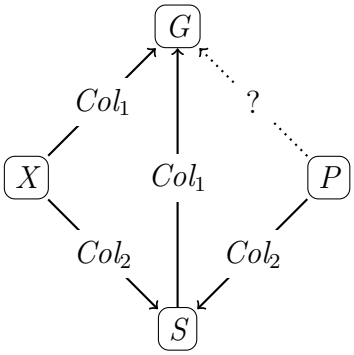
Definition 3. *The* encoding *of a KG $\mathcal{K}$ over* Types *and* Rels*, and a set Λ of candidate triples of the same signature as $\mathcal{K}$ is the δ-annotated graph $G^{\Lambda}_{\mathcal{K}}$ where*

- *$G^{\Lambda}_{\mathcal{K}}$ has a node $u_{c,d}$ for every two constants c, d in $\mathcal{K}$ such that*
  - *either $c = d$,*
  - *or $c \prec d$ and $\mathcal{K} \cup \Lambda$ contains a triple of the form (c, r, d) or (d, r, c) for some relation r;*
- *the feature vectors of nodes in $G^{\Lambda}_{\mathcal{K}}$ are defined as*
  - *$(\mathbf{u}_{c,c})_{\mathsf{id}(t)} = 1$ for all $(c, \mathsf{type}, t) \in \mathcal{K}$,*
  - *$(\mathbf{u}_{c,d})_{\mathsf{id}(r)} = 1$ for all $(c, r, d) \in \mathcal{K}$ with $c \preceq d$ and*
  - *$(\mathbf{u}_{c,d})_{\mathsf{id}(r^-)} = 1$ for all $(d, r, c) \in \mathcal{K}$ with $c \preceq d$,*
  - *all other elements are 0;*
- *$G^{\Lambda}_{\mathcal{K}}$ has an edge between different nodes u_X and u_Y if pairs X and Y have a constant in common.*

The graph in Figure 3.2(a) depicts the structure of $G^{\Lambda}_{\mathcal{K}}$ for our running example KG $\mathcal{K}$ inside the frame in Figure 2.1 and $\Lambda = \{(\mathit{Plato}, \mathsf{lives}, \mathit{Greece})\}$ (where we assume that enumeration id of relations is lexicographic). Nodes represent pairs of constants, (we use abbreviations for succinctness; e.g., (G, X) denotes $u_{\mathit{Greece},\mathit{Xenophon}}$), and two nodes are connected only if their pairs share a constant. Since our example $\mathcal{K}$ has no types and two relations, each feature vector (not shown in the figure for clarity) has dimension 4. For instance, the vector of (G, X) is $[0, 0, 1, 0]$; the first two components are set to 0 because $\mathcal{K}$ does not contain the triple $(\mathit{Greece}, \mathsf{lives}, \mathit{Xenophon})$ or $(\mathit{Greece}, \mathsf{student}, \mathit{Xenophon})$, the third is 1 since $\mathcal{K}$ contains $(\mathit{Xenophon}, \mathsf{lives}, \mathit{Greece})$, and the last is 0 because there is no triple $(\mathit{Xenophon}, \mathsf{student}, \mathit{Greece})$ in $\mathcal{K}$. As we will see in Section 3.3.3, our model predicts the triple $(\mathit{Plato}, \mathsf{lives}, \mathit{Greece})$ to be in the completion $\mathcal{K}^*$ by examining the feature vector of (G, P) after processing $G^{\Lambda}_{\mathcal{K}}$ using a GNN as in Section 3.3.2; in the encoding, the node (G, P) is justified in $G^{\Lambda}_{\mathcal{K}}$ by Λ and initialised with vector $[0, 0, 0, 0]$, since *Plato* and *Greece* are not connected in $\mathcal{K}$.



(a)



(b)

Figure 3.2: Encodings of the KG in the frame in Figure 2.1 with a candidate triple $(Plato, \mathsf{lives}, Greece)$. (a) Encoding of INDIGO, where the feature vectors of the nodes are omitted and the node names are abbreviated; for example (G, X) denotes $u_{Greece,Xenophon}$. (b) Encodings of R-GCN, where the names of constants are abbreviated to their first letter, and Col_1 and Col_2 are colours.

It is interesting to compare our encoding to that of existing approaches. In particular, Figure 3.2(b) depicts the encoding of R-GCN [115] for the same example. The annotated graph now contains one node per constant, and its directed edges are labelled with colours Col_1 and Col_2, representing relations lives and student; thus, the structure of the annotated graph closely mimics that of the KG. In contrast to our approach, however, the feature vectors associated to each node do not bear any connection with the contents of the KG and are typically initialised randomly. After processing this graph by an appropriate GNN, the R-GCN model applies a scoring function to the feature vectors of *Plato* and *Greece* in the output layer of the GNN to predict the presence of the triple in the completion $\mathcal{K}^*$.

Observe that, differently to R-GCN, the size of the encoding graph in our approach can be larger than that of the original KG. Although the size growth can be quadratic in theory, we next argue that it is almost always linear in practice. To this end, let $\mathcal{K}$ be a KG and let Λ be a set of candidate triples of the same signature. By construction, encoding $G^{\Lambda}_{\mathcal{K}}$ has up to $m + |\mathcal{K}| + |\Lambda|$ nodes annotated by δ-vectors and up to $(m + |\mathcal{K}| + |\Lambda|) * n$ edges, where m is the number of constants in $\mathcal{K}$ and n is the average degree of a constant in $\mathcal{K} \cup \Lambda$ among all the constants (here, the *degree* of a constant c is the number of constants d such that c and d appear in the same triple in $\mathcal{K} \cup \Lambda$). The upper bound of the number of nodes is approached only when for each triple (c, r, d) in $\mathcal{K} \cup \Lambda$, there is not a triple (c, r', d) or (d, r', c) in $\mathcal{K} \cup \Lambda$ with $r \neq r'$. In theory, n is bounded by m, which is, however, approached only if every two constants are connected by a triple in $\mathcal{K}$ or Λ. This hardly happens in practice, since real-world KGs are typically sparse. Hence, we can assume that n is small and the encoding size is linear (assuming that δ, which is the number of types and relations, is also small, which may not be true for some special cases such as the biomedical KGs).

3.3.2 The GNN Model

Our approach relies on a GNN $\mathfrak{N}$ to propagate feature information through edges in the annotated graph generated by the pair-wise encoder. Broadly speaking, a GNN is characterised by *aggregation* and *combination* functions for each layer, where the functions usually depend on learnable parameters. Every layer of the GNN updates the feature vector of each node in the graph by first aggregating the previous vectors of the neighbouring nodes using the aggregation function, and then combining the result with the node's previous vector using the combination function. The vectors in the final layer form the GNN's output.

As discussed in Section 2.5, there is a wide range of GNN variants in the literature. In our approach, we utilise a GCN variant [67], which offers a good balance between expressivity and performance. Please note, however, that the choice of GCNs is not crucial for our approach, and GCNs can be easily replaced with any other suitable GNN variant. In our experiments described in Section 4.5.6, we also analyse the effects of replacing GCNs with more advanced GNN variants such as GATs [134] and GINs [161], and find that GCNs outperform the other variants in our case.

We consider a GCN model with $L \in \mathbb{N}$ layers and dimension δ using the propagation rule (2.19) presented in Section 2.5. In particular, we take $\delta^L = \delta^0 = \delta$, and use ReLU as the non-linear activation function σ^ℓ in rule (2.19) for each $\ell < L$ and the logistic sigmoid function for $\ell = L$. The reason why we choose ReLU for the hidden layers is that it is computationally efficient and prevents the vanishing gradient problem. For the output layer, we need a function mapping all the elements to the range $(0, 1)$. In addition, we believe it is possible that there is more than one relation, or even no relation, between two constants. As a consequence, we choose the logistic sigmoid instead of softmax for the output layer so that the outputs are not mutually exclusive.

Observe that the learnable parameters of the GNN (the vector components in the functions) do not depend on the number of nodes in $G^\Lambda_\mathcal{K}$, and are only characterised by annotation dimension δ, which is determined by the number of relations and types that are fixed in the inductive setting. Thus, a GNN is applicable to graphs of any size, and when a trained GNN is applied to a new graph with nodes unseen during training, there is no need to update its parameters. Furthermore, this is also where the heuristics behind the effectiveness of our model lies: the parameters of the GNN effectively captures the relational semantics, that are logic patterns among relations and types. For instance, in the running example of Figure 3.2, the parameters of the GNN captured the interactions between the relations lives and student, and our model successfully learned a pattern using such relations: students and their teachers tend to live in the same country. This pattern is constant-independent, and can thus generalise to any KGs using such relations.

3.3.3 Decoding the GNN Output

The result $\mathfrak{N}(G^\Lambda_\mathcal{K})$ of a GCN $\mathfrak{N}$ applied to the encoding $G^\Lambda_\mathcal{K}$ of $\mathcal{K}$ and a candidate set Λ can be decoded into a set of triples over the same signature by essentially inverting the encoder. Note, however, that features in $\mathfrak{N}(G^\Lambda_\mathcal{K})$ are not Boolean; so, to decide if

the decoded graph contains a triple, we check if the corresponding feature in $\mathfrak{N}(G^{\Lambda}_{\mathcal{K}})$ is above a threshold θ, which we take as 0.5 in our experiments.

Definition 4. *Given a threshold value $\theta \in \mathbb{R}$, the* decoding *of δ-annotated graph G is the set $\mathcal{K}_{\mathsf{dec}}(G)$ of the following triples, for c, d constants, t a type, and r a relation:*

- (c, type, t) *such that* $(\mathbf{u}_{c,c})_{\mathsf{id}(t)} \geq \theta$,
- (c, r, d) *such that* $(\mathbf{u}_{c,d})_{\mathsf{id}(r)} \geq \theta$ *and* $c \preceq d$,
- (d, r, c) *such that* $(\mathbf{u}_{c,d})_{\mathsf{id}(r^-)} \geq \theta$ *and* $c \preceq d$.

Overall, a triple λ in a candidate set Λ for a KG $\mathcal{K}$ is predicted by a GCN $\mathfrak{N}$ to be in the completion $\mathcal{K}^*$ if $\lambda \in \mathcal{K}_{\mathsf{dec}}(\mathfrak{N}(G^{\Lambda}_{\mathcal{K}}))$. Note, however, that in training and evaluation we sometimes need not just a Boolean prediction for a candidate triple, but a *confidence* in this prediction—that is, a numeric value from $(0, 1)$; for this, we directly take the corresponding component of $\mathbf{u}_{c,d}$ in the final layer.

3.3.4 Capturing Logical Rules

As pointed out by Teru et al. [127], the inductive KG completion can also be considered as a *logic induction* problem, which extracts from the KG a set of logic rules that can capture entity-independent relational semantics. In Section 3.3.2, we also highlighted the intuition that in our approach the GNN parameters are able to effectively capture the relational semantics. In this spirit, in addition to using standard benchmarks, we will also compare the inductive capabilities of completion approaches in terms of their ability to capture common inference patterns represented in Datalog.

Following the definition of (Datalog) rules in Section 2.4, in our context, it is convenient to see atoms as triples as in a KG where variables are used instead of constants. Each assignment σ of constants to variables $\mathbf{x}$ of a rule r of form (2.13) maps the atoms $B_1, \ldots, B_n$ into a KG $\mathcal{K}^{\sigma}_r$; in the same way, σ maps H to a triple t^{σ}_r. For instance, consider a rule instantiating the composition pattern

$$(x, \mathsf{Colleague}, y) \wedge (y, \mathsf{Colleague}, z) \rightarrow (x, \mathsf{Colleague}, z), \tag{3.11}$$

and an assignment mapping x to Mary, y to Lily, and z to Lucy. Then, the body atoms are mapped in to a KG $\{(\text{Mary}, \mathsf{Colleague}, \text{Lily}), (\text{Lily}, \mathsf{Colleague}, \text{Lucy})\}$, and the head atom is mapped to a triple $(\text{Mary}, \mathsf{Colleague}, \text{Lucy})$. A completion function (or a GNN-based model realising this function) *captures* r if, for every assignment σ of constants to $\mathbf{x}$, the model predicts triple t^{σ}_r when applied to the KG $\mathcal{K}^{\sigma}_r$.

In general, verifying whether a completion function captures a rule requires checking an infinite number of assignments. We can, however, restrict ourselves to a small

finite number and rely on the following proposition when comparing, in Section 3.5.6, the ability of different approaches to capture rules in practice, provided the completion function f realised by a system under consideration is *constant-agnostic*—that is, such that $f(\mathcal{K}, \lambda) = f(\varrho(\mathcal{K}), \varrho(\lambda))$ for every KG $\mathcal{K}$, candidate triple λ, and renaming ϱ of constants. As far as we could check, all the systems considered in our work realise constant-agnostic completion functions.

Proposition 1. *Let r be a rule of form (2.13), let C be a set of $|\mathbf{x}|$ constants, and let Σ be the set of all assignments of constants from C to $\mathbf{x}$. A constant-agnostic completion function f captures r if and only if $f(\mathcal{K}_r^\sigma, t_r^\sigma)$ is true for each $\sigma \in \Sigma$.*

Proof. The "only if" direction of the claim holds by definition, so we concentrate on the "if" direction. Let f be a constant-agnostic completion function such that $f(\mathcal{K}_r^\sigma, t_r^\sigma)$ is true for each $\sigma \in \Sigma$. To show that f captures r, consider an arbitrary assignment σ' of constants to $\mathbf{x}$ (which is not necessarily in Σ). Fix an arbitrary renaming ϱ of constants in the range of σ' by constants in C, which exists since $|C| = |\mathbf{x}|$. On the one hand, we have

$$f(\mathcal{K}_r^{\sigma'}, t_r^{\sigma'}) = f(\varrho(\mathcal{K}_r^{\sigma'}), \varrho(t_r^{\sigma'})),$$

since f is constant-agnostic. On the other hand,

$$f(\varrho(\mathcal{K}_r^{\sigma'}), \varrho(t_r^{\sigma'})) = f(\mathcal{K}_r^\sigma, t_r^\sigma)$$

by construction, where σ is the substitution from Σ that is the composition of σ' and ϱ. Thus, the claim follows by the assumption that $f(\mathcal{K}_r^\sigma, t_r^\sigma)$ is true. □

3.4 A Discussion on the Pair-Wise Encoding

The key contribution of our approach for inductive KG completion lies in the pair-wise encoder described in Section 3.3.1. In this section, we highlight the insights behind this contribution and explains why it is beneficial to the KG completion task by providing a systematic analysis of current GNN-based methods for KG completion, which can primarily divided into approaches that learn *single*-node representations (or that with *unary* encoders) and those that learn *pairwise*-node representations (or that with *binary/pairwise* encoders). It is worth noting that there has been a body of literature that investigates the theoretical aspects of the GNN-based methods in this manner [9, 60, 178]. However, none of the literature has mentioned our approach, and this section aims to examine how our approach fits into such theoretical analysis.

The GNN-based methods learning *single*-node representations are an extension of GCNs designed to handle relational data. Prominent examples of such methods include R-GCN [115] and CompGCN [132], which are introduced in Section 3.2.3. In general, they use a *unary* encoding function $f_{\mathsf{enc}} : V \to \mathbb{R}^{\delta^L}$ to map each constant in the KG to an individual feature representation, in the sense that the representation of one node is not conditioned on the other nodes. In order to make predictions, they employ a *binary* decoding function $f_{\mathsf{dec}} : \mathbb{R}^{\delta^L} \times \mathbb{R}^{\delta^L} \to \mathbb{R}$, mapping the learnt representations for the pair of nodes in the candidate triple to the predicted score for the candidate triple. The expressive power of these approaches have been extensively discussed in the literature [9, 60], where R-GCN and CompGCN are shown to have limitations in distinguishing nodes in the KGs. In particular, Barcelo et al. [9] characterises the logic expressiveness of these architectures using the multi-relational 1-dimensional Weisfeiler-Leman algorithm (1-RWL). It is shown that these two architectures is upper bounded by 1-RWL for the capacity of R-GCN and CompGCN to distinguish nodes in the KGs; and over every KGs, there is an R-GCN or CompGCN model that is as powerful as 1-RWL. Consider a simplified example depicted in Figure 3.3(a), where there is only one relation R_1 in the graph. In this KG, nodes u and v are isomorphic, and the unary encoders will produce the same representations for them. As a result, R-GCN and CompGCN are not able to distinguish the candidate triples (u, R_1, w) and (v, R_1, w), since they have the same predictions using a binary decoder. From this example, we can see that the approaches learning *single*-node representations are unable to capture the dependency information between nodes in a candidate triple, such as their distance and shared neighbourhoods.

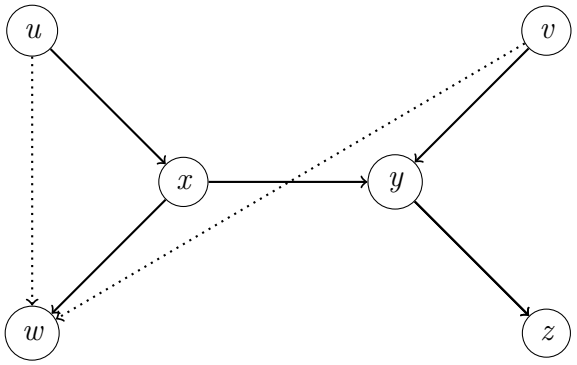
The limitations of unary encoders are addressed by the approaches learning *pairwise*-node representations, such as SEAL [176], GraIL [127], NBFNet [181], and our approach. A key difference of these approaches from R-GCN and CompGCN is that they rely on a *binary* encoding function ${g_{\mathsf{enc}}}^q : V \times V \to \mathbb{R}^{\delta^L}$ conditioned on the candidate triple q, and a *unary* decoding function $g_{\mathsf{dec}} : \mathbb{R}^{\delta^L} \to \mathbb{R}$. For example, SEAL and GraIL first identify an enclosing subgraph for the candidate triple, and initialise the features using a labeling trick [178] based on the the nodes' shortest distance to the target node pair (i, j) of the candidate triple (i, R, j) in the subgraph. They then aggregate information over the enclosing subgraphs, and produce the representations conditioned on the candidate triple. NBFNet is slightly different from SEAL and GraIL with an initialisation conditioning *only* on the source node i, and a message passing mechanism aggregating features through relational paths. As we can see, all these architectures learn pairwise representations relative to the candidate triple

using a binary encoder. Consider the running example in Figure 3.3(a), assume that now (u, R_1, w) is the candidate triple. For these approaches with binary encoders, the representation of pair (u, w) is different from the representation of pair (v, w), since the source node u is labeled differently from the other nodes, including node v. As a consequence, these approaches can distinguish the non-isomorphic facts (u, R_1, w) and (v, R_1, w).

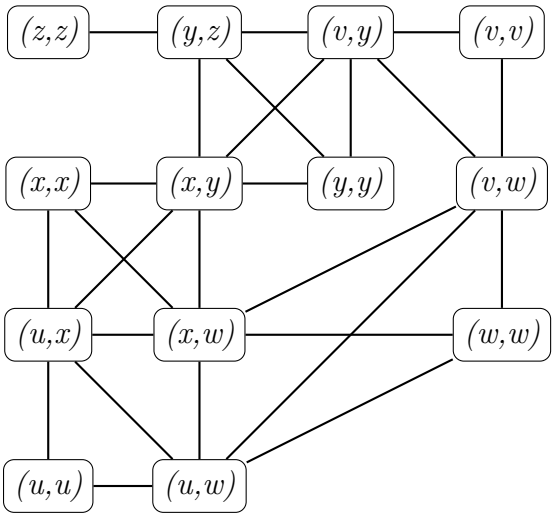
Intuitively, in our pair-wise encoding approach, each node in the annotated graph corresponds to a pair of constants connected by a triple in the KG or by the candidate triple. In this spirit, our approach also learns pairwise representations of constants in the KG conditioned on the candidate triple. A key difference of our approach from the other approaches with binary encoder mentioned above is that, our encoding do not condition on the candidate triple through the feature initialisation (or labeling in Zhang et al. [178]), and our initialisation only relies on the existence of corresponding triples. Instead, our encoding relies on the candidate triple as the structure of the annotated graph varies when the candidate triple changes. Figure 3.3(b) depicts the annotated graph produced by our pair-wise encoder when the candidate set is $\{(u, R_1, w), (v, R_1, w)\}$. As we can see, the nodes (u, w) and (v, w) in the annotated graph are not isomorphic, even though they are both initialised with a feature vector [0, 0]. Consequently, the node representations learnt by the GNNs are different, which will result in different predictions. Hence, thanks to the pair-wise encoder, our approach can distinguish some structures that R-GCN and CompGCN can not distinguish. The discussion about unary and binary encoders and the running examples can help reveal the insights of our approach to some extent. There have been some preliminary theoretical results about the logic expressiveness of prominent approaches such as SEAL and NBFNet. For example, Huang et al. aligned architectures including NBFNet with the relational asymmetric local 2-WL (rawl_2), and illustrated the binary classifiers that can be expressed as these architectures. We refer the readers to Huang et al. [60] and Zhang et al. [178] for details and formal proofs. We believe such characterisation can also be extended to our approach, and we leave this for future work.

3.5 Evaluation

We have implemented our approach using Python and PyTorch v1.4.0 in a system called INDIGO. In this section, we will evaluate INDIGO and baselines including R-GCN [115], GraIL [127], and the system by Hamaguchi et al. [50], on a variety of



(a)



(b)

Figure 3.3: Examples for unary and binary encodings. (a) A graph for which R-GCN and CompGCN can not distinguish (u, R_1, w) and (v, R_1, w). (b) Encodings of INDIGO, where the nodes (u, w) and (v, w) are not isomorphic.

Table 3.1: Benchmarks statistics, where $|\mathcal{P}_{\text{train}}|$, $|\mathcal{P}_{\text{valid}}|$, $|\mathcal{K}_{\text{test}}|$, and $|\mathcal{P}_{\text{test}}|$ are the sizes of the corresponding training set, validation set, incomplete test KG, and positive test triple set, respectively.

	GraIL-BM / FB15K-237				GraIL-BM / NELL-995				GraIL-BM / WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
$\lvert\mathcal{P}_{\text{train}}\rvert$	4,245	9,739	17,986	27,203	4,687	8,219	16,393	7,546	5,410	15,262	25,901	7,940
$\lvert\mathcal{P}_{\text{valid}}\rvert$	489	1,166	2,194	3,352	414	922	1,851	876	630	1,838	3,097	934
$\lvert\mathcal{K}_{\text{test}}\rvert$	1,993	4,145	7,406	11,714	833	4,586	8,048	7,073	1,618	4,011	6,327	12,334
$\lvert\mathcal{P}_{\text{test}}\rvert$	205	478	865	1,424	100	476	809	731	188	441	605	1,429

	Hamaguchi-BM									INDIGO-BM
	h-1K	h-3K	h-5K	t-1K	t-3K	t-5K	b-1K	b-3K	b-5K	
$\lvert\mathcal{P}_{\text{train}}\rvert$	108,197	99,963	92,309	96,968	78,763	67,774	93,364	71,097	57,601	121,601
$\lvert\mathcal{P}_{\text{valid}}\rvert$	4,613	4,184	3,845	3,999	3,122	2,601	3,799	2,759	2,166	14,121
$\lvert\mathcal{K}_{\text{test}}\rvert$	4,352	12,376	19,625	15,277	31,770	40,584	18,638	38,285	48,425	250,195
$\lvert\mathcal{P}_{\text{test}}\rvert$	994	2,969	4,919	986	2,880	4,603	960	2,708	4,196	14,904

benchmarks. The rest of this section is organised as follows. In Section 3.5.1, we will introduce the details of benchmarks used in our evaluation. Then, we will describe the performance metrics in Section 3.5.2, and the training strategy in Section 3.5.3. Furthermore, the evaluation results will be discussed in 3.5.4, and an ablation study will be presented in 3.5.5. Finally, we will show the results of experiments for the systems' ability to learn common inference patterns in Section 3.5.6. All experiments were performed on an Intel(R) Xeon(R) machine with 8 cores and a 2.6 GHz CPU equipped with 540 GB of RAM running Fedora 33 (x86_64). We also tried with a Macbook Pro 2021 with 10-core CPU, 24-core GPU and 32 GB memory, and it showed that all the experiments can also be completed on such device. The code, benchmarks, and the accompanying documentation, are available online.[1]

3.5.1 Benchmarks

We exploit a number of benchmarks proposed by Teru et al. [127] and Hamaguchi et al. [50] for inductive KG completion. The benchmarks by Teru et al., 12 in total, are based on transductive benchmarks FB15K-237 [128], NELL-995 [159], and WN18RR [36], and have four versions for each of these transductive benchmarks with increasing sizes; we will call them using the pattern GraIL-BM / *XXX*.v*i* where *XXX* is the base and i the number of the version (e.g., GraIL-BM / NELL-995.v3 is

[1] https://github.com/shuwen-liu-ox/INDIGO

the third version of the benchmark based on NELL-995). Similarly, the benchmarks by Hamaguchi et al., 9 in total, are all based on a transductive benchmark WordNet11 [120], and we call them Hamaguchi-BM / X-iK, where X is one of h, t, b and $i = 1, 3, 5$ (these parameters specify how the benchmark was constructed, and we refer the readers to the original literature for the procedure of the benchmark construction [50, 127]).

As discussed in Section 2.2, each of these benchmarks provides the following:

- disjoint sets $\mathcal{P}_{\text{train}}$ and $\mathcal{P}_{\text{valid}}$ of triples with $\mathsf{Sig}(\mathcal{P}_{\text{valid}}) \subseteq \mathcal{P}_{\text{train}}$ for training and validation;
- an incomplete KG $\mathcal{K}_{\text{test}}$ and a set $\mathcal{P}_{\text{test}}$ of test triples with $\mathsf{Sig}(\mathcal{P}_{\text{test}}) \subseteq \mathsf{Sig}(\mathcal{K}_{\mathsf{test}})$ that are to hold in the completion of $\mathcal{K}_{\mathsf{test}}$ for testing.

These benchmarks implicitly assume that all provided triples represent true facts. In contrast, all other triples over the relevant signature represent facts that are unknown, but assumed to be *pseudo-negative*—that is, are false with equal probability. This relatively weak assumption is due to the fundamental incompleteness and open-world nature of existing KGs, which make it difficult to discern the truth status of a triple not mentioned in the KG when designing a benchmark. The equal probability ensures that no system has an advantage when evaluated on these benchmarks under the assumption that all pseudo-negative triples are truly negative. Furthermore, to capture the inductive setting, triples in all sets use the same types and relations, but the test sets contain constants that are not mentioned in the training and validation sets.

These existing benchmarks are, however, limited in that they capture the KG evolution scenario in Figure 2.1 only partially. On the one hand, in benchmarks GraIL-BM the training and test sets mention completely disjoint sets of constants, thus not capturing the situation where triples mentioning both existing and new constants are added to the KG; on the other hand, in benchmarks Hamaguchi-BM each triple in the testing set always uses both a constant mentioned in the training set and an unseen constant, thus not capturing the situation where a KG is also extended with triples mentioning only unseen constants.

To address this limitation, we have designed a new benchmark, called INDIGO-BM, which is based on FB15K-237 and has the same structure and assumptions as the existing benchmarks, but where the use of unseen constants in the test sets is not restricted in any specific way. Additionally, in contrast to existing benchmarks, which contain no type information, our benchmark comes equipped with type triples.

Our INDIGO-BM benchmark was constructed in the following way. We first collected from Freebase all the type triples involving any constant contained in FB15K-237, and merged them with the triples in FB15K-237, which resulted in a KG $\mathcal{K}$. Then, we randomly sampled $1,000$ triples from FB15K-237, and marked all the constants mentioned in these triples as *unseen constants*. Third, for the triples in $\mathcal{K}$ that contains no unseen constants, we split them into a training set $\mathcal{P}_{\text{train}}$ and a validation set $\mathcal{P}_{\text{valid}}$ with a ratio of 9:1. Finally, the triples in $\mathcal{K}$ that contains at least one unseen constant were split into an incomplete graph $\mathcal{K}^{-}_{\text{test}}$ and a set $\mathcal{P}_{\text{test}}$ of positive test triples with a ratio of 9:1. To simulate a KG evolution scenario, we took the incomplete KG $\mathcal{K}_{\mathsf{test}} = \mathcal{K}^{-}_{\text{test}} \cup \mathcal{P}_{\text{train}}$.

The statistics of all our 22 benchmarks are summarised in Table 3.1.

3.5.2 Performance Metrics

We evaluate the systems' performance using standard classification metrics and ranking metrics introduced in Section 2.3, which are computed based on the systems' outcomes on sets P_{test} and N_{test} of positive and negative examples defined in Section 2.3. For a benchmark with incomplete KG $\mathcal{K}_{\text{test}}$ and a set $\mathcal{P}_{\text{test}}$ of test triples, we obtain N_{test} by sampling from the set N^{*}_{test} of all pairs $(\mathcal{K}_{\text{test}}, \lambda)$ with the pseudo-negative triples $\lambda \notin \mathcal{K}_{\text{test}} \cup \mathcal{P}_{\text{test}}$ and $\mathsf{Sig}(\lambda) \subseteq \mathsf{Sig}(\mathcal{K}_{\text{test}})$ using different sampling methods for classification and ranking-based metrics as described below. Note that sampling is necessary because N^{*}_{test} is usually very large, and so using all its triples is infeasible. To mitigate the effects of possible fluctuations caused by sampling, we evaluate each system on a given benchmark over 10 runs with independently sampled sets of negative examples, and report the average and variance for each metric.

For classification-based metrics, we construct N_{test} by randomly sampling, with equal probability, one element of N^{*}_{test} for each positive example in P_{test}. This method ensures that systems cannot gain advantage by adopting a particular sampling strategy for negative examples during training [104]. In our experiments, we use precision, recall, accuracy, F1-score, and AUC-PR for the classification-based metrics. The details of how they are computed are described in Section 2.3.1.

For *ranking-based* metrics, we construct $N_{\text{test}} = \bigcup_{p \in P_{\text{test}}} N^{\mathsf{s}}_{p} \cup N^{\mathsf{R}}_{p} \cup N^{\mathsf{o}}_{p}$ following the way described in Section 2.3.2, except that each of N^{s}_{p} and N^{o}_{p} contains only 50 randomly sampled negative examples instead of all possible negative examples. We then compute the ranking-based metrics C-Hits@k, R-Hits@k, C-MRR, and R-MRR following the way described in Section 2.3.2.

3.5.3 Training

Our INDIGO system is trained as a *denoising autoencoder* [135], which can help extract robust and informative representations from corrupted data. The training set $\mathcal{P}_{\text{train}}$ of a benchmark is first randomly split, with ratio 9:1, into an incomplete KG $\mathcal{K}_{\text{train}}$ and a set $\mathcal{T}_{\text{train}}$ of triples assumed to hold in the completion of $\mathcal{K}_{\text{train}}$. Then, the model is trained on positive examples $(\mathcal{K}_{\text{train}}, \lambda)$ for each $\lambda \in \mathcal{T}_{\text{train}}$ and negative examples $(\mathcal{K}_{\text{train}}, \lambda)$ for λ sampled from the set $\mathcal{N}$ of triples not in $\mathcal{T}_{\text{train}}$ using the strategy described next. For each triple $(c, r, d) \in \mathcal{T}_{\text{train}}$ with $r \neq \mathsf{type}$ we sampled (with equal probability) the following triples from $\mathcal{N}$: three triples of the form (c, r', d) where r' is *disjoint* from r role in $\mathcal{P}_{\text{train}}$—that is, such that $\mathcal{P}_{\text{train}}$ has no triples (c', r, d') and (c', r', d'); three triples of the form (c, r, d'), if r is *functional* in $\mathcal{P}_{\text{train}}$—that is, $\mathcal{P}_{\text{train}}$ has no triples (c', r, d_1), (c', r, d_2) with $d_1 \neq d_2$; and three triples of the form (c', r, d), if r is *inverse-functional* in $\mathcal{P}_{\text{train}}$—that is, $\mathcal{P}_{\text{train}}$ has no triples (c_1, r, d'), (c_2, r, d') with $c_1 \neq c_2$. For each triple $(c, \mathsf{type}, t) \in \Lambda^+_{\mathsf{train}}$ we similarly sampled from $\mathcal{N}$ three triples of the form (c, type, t') where t' is *disjoint* from t type in $\mathcal{P}_{\text{train}}$, which is defined analogously to disjoint roles. Please note that here we generate negative examples from a pure data perspective, and it is likely that the negative example still holds in the semantic sense. For instance, if type t is subsumed by type t', then (c, type, t') should still hold for $(c, \mathsf{type}, t) \in \Lambda^+_{\mathsf{train}}$ even if t' is disjoint from t in $\mathcal{P}_{\text{train}}$. In the current setting, we assume such information about types is not available; if the information becomes available, we can add one more constraint that t should not be subsumed by t'.

As a result, we generate up to 9 negative training examples for each positive example.

We employed the standard Cross-Entropy (CE) loss function and trained for 3,000 epochs using Adam optimisation with L2 penalty 5e-8. We set as hyper-parameters the number of layers in the GCN (2, 3, or 4), the dimension of vectors in the hidden layers (32, 64, or 128) and the learning rate (0.01 or 0.001) and cross-validated them on each of the benchmarks using the validation sets to obtain a most favourable (for all benchmarks) setting of 2 layers, vector dimension of 64, and learning rate 0.001.

Our baseline systems are trained similarly as autoencoders. We trained them using their (publicly available) code and the settings reported in the papers without modifications.

Table 3.2: Classification-based metric results on the GraIL-BM and INDIGO-BM benchmarks in %

		GraIL-BM / FB15K-237				GraIL-BM / NELL-995				GraIL-BM / WN18RR				INDIGO-BM
		v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4	
Prec.	R	51.1	51.5	54.0	51.4	29.3	51.8	52.1	52.7	50.2	52.9	51.2	48.5	66.8
	G	41.5	62.7	63.9	63.7	**96.4**	38.7	49.3	61.1	79.0	62.6	54.8	74.0	77.5
	I	**92.2**	**95.7**	**95.1**	**93.8**	78.2	**94.6**	**95.6**	**85.8**	**79.4**	**82.0**	**85.7**	**82.1**	**98.1**
Rec.	R	47.3	44.4	66.8	77.9	93.8	57.6	56.9	71.3	62.2	50.1	**94.9**	49.8	**95.0**
	G	**92.2**	**95.8**	**97.0**	**92.6**	98.2	**95.6**	**98.7**	49.7	**98.0**	**99.6**	94.1	**98.3**	93.9
	I	74.9	82.4	82.2	81.0	**99.0**	72.4	83.3	**84.5**	96.7	91.7	82.3	90.8	90.5
F1	R	49.1	47.6	59.7	61.9	44.6	54.5	54.4	60.6	55.6	51.5	66.5	49.1	78.4
	G	57.1	75.8	77.0	75.5	97.3	55.1	65.8	54.8	87.5	76.9	69.3	84.5	84.9
	I	**82.7**	**88.5**	**88.2**	**86.9**	**87.3**	**82.0**	**89.0**	**85.1**	**87.2**	**86.3**	**84.0**	**86.2**	**94.1**
Acc.	R	51.0	51.3	54.9	52.1	63.7	52.0	52.3	53.6	50.2	52.7	52.2	48.4	73.9
	G	69.0	80.0	80.1	79.3	**97.3**	68.5	74.3	49.7	**88.7**	81.2	75.7	**86.4**	86.2
	I	**84.3**	**89.3**	**89.0**	**87.8**	85.6	**84.1**	**89.7**	**85.2**	85.7	**85.8**	**84.3**	85.4	**94.4**
AUC	R	51.0	50.5	50.5	52.6	74.5	50.4	52.0	51.0	74.5	50.4	52.0	51.0	89.5
	G	78.6	90.0	93.1	89.5	**98.8**	89.7	**95.4**	65.8	**92.3**	**92.7**	82.8	94.4	94.2
	I	**93.4**	**96.3**	**96.6**	**95.8**	94.5	**92.5**	95.1	**92.9**	91.2	92.5	**92.4**	**94.7**	**99.0**

3.5.4 Evaluation Results

The results for classification-based metrics of the metric-base evaluation of the systems on the benchmarks are summarised in Table 3.2 and Table 3.3 (the system by Hamaguchi et al. is not applicable to GraIL-BM and INDIGO-BM due to its limited inductive capabilities). The results for ranking-based metrics are summarised in Table 3.4, Table 3.5, Table 3.6, and Table 3.7. In these tables, 'R', 'G', 'H', and 'I' stand for R-GCN, GraIL, the system by Hamaguchi et al., and INDIGO, respectively. Moreover, 'Prec.', 'Rec.', 'Acc.' stand for precision, recall, and accuracy, respectively.

As we can see, INDIGO consistently outperforms, often by a significant margin, the baselines on almost all metrics. A notable exception is ranking-based entity metrics C-Hits@k and C-MRR, where INDIGO is often worse than the baselines. This can be explained by the fact that the baselines use entity corruption in negative sampling for training, which can be seen as a bias towards these metrics, while our sampling strategy in training is not prejudiced to any metric. In Table 3.13, we also report the variance of each metric. Recall that, when computing R-Hits@k and R-MRR, we took all possible relevant samples for constructing negative examples; so

Table 3.3: Classification-based metric results on the Hamaguchi-BM benchmarks in %

		Hamaguchi-BM / h			Hamaguchi-BM / t			Hamaguchi-BM / b		
		1k	3k	5k	1k	3k	5k	1k	3k	5k
Prec.	R	45.0	45.8	47.5	49.8	44.8	43.7	29.7	33.5	34.0
	G	36.9	37.9	38.4	45.3	44.4	44.4	46.5	46.2	43.2
	H	**82.4**	77.1	77.7	79.0	76.0	73.3	**83.3**	75.3	71.1
	I	75.7	**82.6**	**82.2**	**81.3**	**86.1**	**86.0**	81.9	**83.6**	**86.9**
Rec.	R	63.1	64.5	65.5	64.8	64.7	63.3	40.5	46.0	46.0
	G	49.1	50.3	50.1	54.2	53.8	54.2	54.8	54.5	53.3
	H	**85.6**	**82.7**	83.4	73.8	73.9	75.7	87.7	87.3	86.9
	I	74.9	78.1	**85.2**	**82.5**	**86.0**	**88.6**	**95.2**	**91.8**	**91.6**
F1	R	52.5	53.6	55.0	56.4	53.0	51.7	34.3	38.7	39.1
	G	42.1	43.2	43.5	49.4	48.6	48.8	50.3	50.0	47.7
	H	**84.0**	79.8	80.5	76.3	74.9	74.5	87.7	87.3	86.9
	I	75.2	**80.3**	**83.7**	**82.0**	**85.9**	**87.3**	**88.0**	**87.5**	**89.2**
Acc.	R	43.0	44.1	46.5	49.8	42.5	40.8	22.4	27.3	28.3
	G	49.3	50.2	50.0	53.5	53.1	53.4	54.0	53.8	52.6
	H	**83.6**	79.0	79.7	77.1	75.3	74.0	85.0	79.3	75.8
	I	75.3	**80.8**	**83.3**	**81.8**	**85.8**	**87.0**	**87.0**	**86.8**	**88.8**
AUC	R	43.0	43.0	45.4	48.7	41.4	39.2	33.4	35.4	35.1
	G	51.5	55.6	56.8	58.8	61.7	62.0	59.2	61.1	58.8
	H	77.7	72.4	73.1	71.4	69.2	67.7	79.2	72.1	68.4
	I	**84.6**	**88.7**	**91.4**	**90.3**	**94.3**	**95.3**	**95.4**	**95.7**	**96.8**

Table 3.4: Relation-related ranking-based metric results on the GraIL-BM and INDIGO-BM benchmarks in %

R-		GraIL-BM / FB15K-237				GraIL-BM / NELL-995				GraIL-BM / WN18RR				INDIGO-BM
		v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4	
Hits@1	R	1.4	1.1	0.6	1.5	**16.0**	0.2	0.5	2.1	0.5	6.6	15.5	3.1	17.6
	G	0.5	0.2	1.8	0.8	0.0	2.7	0.7	0.0	0.5	2.0	3.5	16.3	3.4
	I	**36.4**	**42.0**	**48.1**	**42.0**	13.0	**46.5**	**43.4**	**40.1**	**67.5**	**51.7**	**64.2**	**37.0**	**54.1**
Hits@3	R	2.4	3.4	3.5	3.3	26.0	0.8	1.4	3	2.1	11.0	24.5	8.1	32.4
	G	1.0	0.4	6.6	3.0	0.0	7.4	2.5	0.5	0.6	10.7	17.5	22.6	4.9
	I	**53.1**	**67.6**	**66.5**	**66.3**	**80.0**	**56.9**	**64.4**	**45.7**	**98.4**	**97.3**	**91.9**	**96.1**	**77.0**
Hits@10	R	4.4	6.7	6.5	6.6	92.0	4.8	8.5	10.0	**100.0**	**100.0**	95.9	**100.0**	54.1
	G	6.3	4.6	20.7	10.1	87.0	17.2	10.1	7.3	**100.0**	**100.0**	98.3	**100.0**	10.2
	I	**75.2**	**85.3**	**84.1**	**83.0**	**100.0**	**80.3**	**81.2**	**81.7**	**100.0**	**100.0**	**99.8**	**100.0**	**92.4**
MRR	R	4.0	4.3	3.8	4.2	32.5	3.4	3.5	6.1	16.1	20.7	30.6	19.4	29.3
	G	3.5	2.6	8.5	5.0	14.5	9.6	4.7	3.9	17.6	23.0	22.4	29.8	6.4
	I	**48.9**	**57.1**	**59.8**	**56.4**	**50.0**	**56.2**	**56.2**	**50.8**	**82.8**	**74.2**	**78.2**	**66.2**	**67.8**

Table 3.5: Relation-related ranking-based metric results on Hamaguchi-BM benchmarks in %

R-		Hamaguchi-BM / h			Hamaguchi-BM / t			Hamaguchi-BM / b		
		1k	3k	5k	1k	3k	5k	1k	3k	5k
Hits@1	R	7.9	7.8	9.1	7.2	10.0	6.6	7.4	5.7	6.2
	G	0.2	1.0	0.8	0.2	0.3	0.7	0.2	0.3	1.2
	H	19.6	14.4	11.8	14.8	12.1	12.7	13.2	14.6	10.8
	I	**45.4**	**44.7**	**38.4**	**36.3**	**45.0**	**42.7**	**50.2**	**43.1**	**49.7**
Hits@3	R	31.4	27.8	31.3	27.6	30.3	25.4	19.8	20.2	22.1
	G	29.6	29.1	26.1	31.7	27.9	25.5	32.1	28.6	27.2
	H	47.1	41.6	38.5	43.3	35.3	37.4	38.5	41.5	34.0
	I	**80.0**	**83.8**	**86.2**	**85.4**	**88.0**	**88.3**	**93.8**	**90.1**	**92.0**
Hits@10	R	94.1	95.3	95.1	96.6	94.3	93.6	96.2	95.4	96.0
	G	97.0	97.8	89.7	97.3	97.2	86.3	96.6	88.6	88.8
	H	97.8	98.2	98.8	97.6	98.4	98.1	98.7	97.1	97.3
	I	**100.0**	**99.8**	**99.9**	**100.0**	**100.0**	**99.9**	**100.0**	**99.8**	**100.0**
MRR	R	28.6	27.6	29.0	26.6	29.1	26.0	25.8	24.6	25.4
	G	20.5	22.0	21.1	21.7	21.3	20.7	21.9	20.4	20.8
	H	39.3	35.5	33.4	36.1	32.4	33.9	33.7	35.6	31.0
	I	**64.7**	**65.2**	**63.1**	**62.5**	**67.6**	**66.6**	**72.1**	**67.4**	**71.3**

Table 3.6: Constant-related ranking-based metric results on the GraIL-BM and INDIGO-BM benchmarks in %

C-		GraIL-BM / FB15K-237				GraIL-BM / NELL-995				GraIL-BM / WN18RR				INDIGO-BM
		v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4	
Hits@1	R	7.3	7.6	4.4	5.8	3.0	3.7	7.2	3.0	6.1	7.1	6.2	2.7	18.0
	G	**34.4**	**48.5**	**56.3**	**44.4**	**49.0**	**53.3**	**65.0**	**51.8**	**71.3**	**78.1**	**50.3**	**74.7**	**53.7**
	I	32.7	25.8	46.4	26.1	35.5	29.6	35.5	34.3	3.5	8.2	21.9	9.5	43.1
Hits@3	R	16.1	18.3	14.1	16.1	7.5	12.2	15.5	9.3	20.1	18.1	16.9	8.8	36.1
	G	43.4	**68.5**	**71.2**	**61.8**	**51.0**	**76.5**	**84.4**	**56.0**	**82.7**	**81.5**	**55.5**	**76.3**	**73.6**
	I	**45.1**	36.2	33.9	37.1	39.5	44.2	45.0	52.3	12.5	18.8	33.1	13.5	53.5
Hits@10	R	47.3	52.2	46.7	49.1	22.1	40.1	46.7	31.7	72.1	65.4	61.2	29.0	64.5
	G	55.4	**82.9**	**85.0**	**84.5**	**57.0**	**90.8**	**93.6**	57.3	**84.0**	**81.6**	**85.4**	**76.3**	**78.8**
	I	**55.6**	48.9	46.9	49.7	44.0	56.6	56.1	**67.8**	30.3	41.2	55.5	25.0	61.2
MRR	R	19.1	20.4	16.8	18.5	10.3	15.3	18.7	12.6	22.2	21.8	20.0	12.0	29.3
	G	**42.0**	**60.8**	**66.1**	**56.9**	**52.6**	**66.9**	**75.8**	**55.4**	**77.3**	**80.2**	**54.2**	**76.2**	**63.3**
	I	41.6	34.6	32.9	35.0	40.2	39.8	43.3	46.3	13.0	18.9	32.4	16.0	50.3

Table 3.7: Constant-related ranking-based metric results on the Hamaguchi-BM benchmarks in %

C-		Hamaguchi-BM / h			Hamaguchi-BM / t			Hamaguchi-BM / b		
		1k	3k	5k	1k	3k	5k	1k	3k	5k
Hits@1	R	20.7	15.8	10.6	9.1	9.4	7.8	30.8	12.7	8.5
	G	4.9	8.8	11.8	14.8	14.0	17.0	17.0	12.7	15.5
	H	**33.1**	**28.5**	**30.1**	**25.8**	17.9	18.2	28.9	21.3	17.0
	I	21.1	22.2	21.8	22.7	**24.9**	**27.5**	**32.5**	**29.5**	**29.3**
Hits@3	R	31.6	29.7	23.9	17.1	24.9	21.3	**62.5**	36.0	24.3
	G	13.8	17.7	21.0	23.9	19.5	26.7	26.5	22.3	26.3
	H	**55.6**	**48.1**	**49.7**	**42.9**	31.6	31.6	46.7	37.1	30.6
	I	32.9	35.3	33.4	31.0	**33.9**	**36.1**	43.2	**37.5**	**39.9**
Hits@10	R	44.7	45.8	42.0	34.7	46.1	50.8	**69.8**	48.2	31.5
	G	44.0	49.3	50.6	50.8	31.9	50.6	32.9	31.0	39.1
	H	**78.5**	**72.9**	**73.4**	**68.4**	**56.9**	**56.2**	58.3	46.6	44.0
	I	49.7	52.6	50.3	42.4	45.8	49.4	57.1	**50.4**	**51.9**
MRR	R	29.7	26.7	21.8	18.1	21.9	20.6	**49.7**	31.4	24.3
	G	14.8	18.8	21.6	24.1	20.7	26.3	26.7	21.6	25.6
	H	**48.2**	**42.8**	**44.1**	**39.3**	30.1	30.2	44.0	34.8	29.7
	I	31.2	32.8	31.7	30.2	**32.8**	**35.7**	40.5	**37.4**	**37.8**

Table 3.8: Training and testing time on GraIL-BM (in hours)

	Model	GraIL-BM / FB15K-237 v1	v2	v3	v4	GraIL-BM / NELL-995 v1	v2	v3	v4	GraIL-BM / WN18RR v1	v2	v3	v4
Train	R-GCN	**1.11**	**2.61**	6.97	17.33	1.41	3.34	6.78	3.50	0.56	1.13	2.59	0.83
	GraIL	6.63	47.11	334.83	471.97	1.92	31.32	254.15	172.96	0.72	1.55	2.98	1.68
	INDIGO	1.31	3.13	**6.64**	**9.95**	**1.21**	**2.34**	**4.83**	**2.25**	**0.32**	**0.66**	**0.89**	**0.45**
Test	R-GCN	**0.002**	**0.004**	**0.01**	**0.02**	**0.004**	**0.01**	**0.02**	**0.02**	**0.003**	**0.01**	**0.02**	**0.03**
	GraIL	0.96	2.95	9.24	12.97	0.05	1.01	3.73	1.22	0.08	0.12	0.18	0.28
	INDIGO	0.04	0.13	0.29	0.44	0.03	0.12	0.26	0.24	0.02	0.04	0.11	0.11

Table 3.9: Training and testing time on Hamaguchi-BM and INDIGO-BM (in hours)

	Model	Hamaguchi-BM / h 1k	3k	5k	Hamaguchi-BM / t 1k	3k	5k	Hamaguchi-BM / b 1k	3k	5k	INDIGO-BM
Train	R-GCN	1.76	1.31	1.21	1.18	1.09	0.90	1.23	0.78	0.55	45.83
	GraIL	9.38	7.72	7.47	8.59	5.87	3.75	9.52	3.68	4.52	69.59
	Hamaguchi et al.	1.33	1.40	1.08	1.07	1.09	0.94	1.54	1.01	0.92	-
	INDIGO	**1.14**	**1.06**	**0.95**	**0.79**	**0.48**	**0.41**	**0.74**	**0.45**	**0.37**	**39.68**
Test	R-GCN	**0.02**	**0.06**	0.12	**0.03**	**0.10**	**0.17**	**0.04**	**0.12**	**0.18**	**2.46**
	GraIL	0.08	0.34	0.44	0.09	0.31	0.56	0.10	0.33	0.59	49.65
	Hamaguchi et al.	0.07	0.21	0.35	0.08	0.22	0.37	0.08	0.21	0.35	-
	INDIGO	**0.02**	**0.06**	**0.11**	0.09	0.18	0.21	0.10	0.19	0.20	3.56

the variance is not applicable in these cases and we omitted it in the table. As we can see, the variance was very small for all the metrics on the benchmarks.

In addition to achieving better results, our system is also significantly faster to train and test. Detailed time statistics for training and testing are given in Table 3.8 and Table 3.9. Compared with Hamaguchi et al. and GraIL, INDIGO takes less time to both train and test; R-GCN is also slower that INDIGO in testing but comparable in training (note that testing time is much smaller that training and thus should not be generally considered as a limiting factor for a system).

Moreover, we also demonstrate in Table 3.10 the size growth of our pair-wise encoding in practice. As we can see, for all the benchmarks we used, the size of the annotated graphs generated by the pair-wise encoder of INDIGO is still reasonable, without experiencing a size explosion. However, please note that this size-growth should not be ignored, and it is crucial to monitor the size growth and even optimise it more efficiently in the future.

Table 3.10: Size of INDIGO Encoding for the Benchmarks. Where $|V|$ and $|E|$ are the number of nodes and edges in the annotated graph generated by INDIGO using the training set of each benchmark, respectively.

	GraIL-BM / FB15K-237				GraIL-BM / NELL-995			
	v1	v2	v3	v4	v1	v2	v3	v4
$\|V\|$	10,283	18,254	31,641	43,342	8,321	11,647	21,997	10,519
$\|E\|$	671,549	1,593,090	3,665,293	5,934,924	324,651	1,345,859	3,483,141	1,550,511

	GraIL-BM / WN18RR				INDIGO-BM
	v1	v2	v3	v4	
$\|V\|$	7,177	19,385	33,723	10,440	141,490
$\|E\|$	63,301	194,169	403,141	98,826	13,095,500

	Hamaguchi-BM								
	h-1K	h-3K	h-5K	t-1K	t-3K	t-5K	b-1K	b-3K	b-5K
$\|V\|$	95,492	90,042	84,674	88,328	75,645	67,248	85,880	69,913	59,260
$\|E\|$	2,831,262	2,522,204	2,131,054	1,788,408	772,501	595,250	1,714,266	653,041	462,638

Table 3.11: Ablation results on GRAIL-BM/WN18RR.v1 in %

	GNN	F1-Score	Accuracy	AUC-PR	R-MRR	C-MRR
INDIGO	GCN	**87.2**	**85.7**	**91.2**	**82.8**	13.0
INDIGO	GAT	86.7	85.3	89.4	64.8	**15.6**
INDIGO	GIN	86.9	85.5	90.1	73.9	12.5

3.5.5 Ablation Studies

We have also conducted ablation experiments to analyse the significance of components in our approach. As already discussed, the encoder and decoder constitute the key novelty of INDIGO; replacing them with a more conventional encoding approach would result in a GNN-based KG completion system similar in structure to R-GCN. In Section 3.5.4, we have already demonstrated the superior performance of INDIGO over R-GCN. Thus, we analysed the consequences of replacing the GCN used in INDIGO with a GAT, which exploits attention to learn different weights to different neighbourhood; and a GIN, whose distinguishing power coincides with that of the Weisfeiler-Lehman graph isomorphism test.

The results are shown in Table 3.11. As we can see, INDIGO with GCN outperform the variants with GIN or GAT. A possible explanation is that our approach benefits from a simple architecture of GCNs since the encoder is already capturing the relevant structural information in an explicit and transparent way; in contrast, the more sophisticated GNN architectures can introduce noise.

3.5.6 Capturing Logical Rules

We also studied the systems' ability to learn rules by means of capturing, as discussed in Section 3.3.4. In particular, we considered several commonly used rule patterns, which are specified in the first column of Table 3.12. Each pattern represents all rules obtained by appropriately substituting its templates $\_\mathsf{a}$ by types and relations. We took GRAIL-BM/NELL-995.v3 and INDIGO-BM benchmarks for experiments, and considered INDIGO, GraIL, and R-GCN trained for these benchmarks. For each benchmark with an incomplete KG $\mathcal{K}_{\text{test}}$ and a set $\mathcal{P}_{\text{test}}$ of test triples, and for each rule pattern we proceeded as follows. We first generated all rules represented by the pattern over the types and relations in $\mathsf{Sig}(\mathcal{K}_{\text{test}})$. We then identified the rules with confidence at least 0.7, where the *confidence* of a rule r in $\mathcal{K}_{\text{test}}$ and $\mathcal{P}_{\text{test}}$ is n/m for m the number of assignments σ such that $K_r^\sigma \subseteq \mathcal{K}_{\mathsf{test}}$ and n the number of such σ that also satisfy $t_r^\sigma \in \mathcal{K}_{\mathsf{test}} \cup \mathcal{P}_{\text{test}}$. Finally, for each model we computed, using Proposition 1, the proportion of captured high-confidence rules.

Our results are summarised in Table 3.12. As we can see, INDIGO was consistently able to capture the highest number of high-confidence rules for each of the patterns. This suggests that our system is able to inductively generalise its predictions more effectively than the baselines.

Table 3.12: Results on capturing high-confidence rules, where each entry indicates both the number of the high-confidence rules of the benchmark represented by the pattern that are captured by the model and the percentage of the total number of the represented rules.

	GraIL-BM / NELL-995.v3		
Rule pattern	R-GCN	GraIL	INDIGO
$(x, \_\mathtt{r}, y) \rightarrow (x, \_\mathtt{s}, y)$	-	-	-
$(x, \mathsf{type}, \_\mathtt{t}) \rightarrow (x, \mathsf{type}, \_\mathtt{u})$	-	-	-
$(x, \_\mathtt{r}, y) \rightarrow (y, \_\mathtt{r}, x)$	0	0	**4 (80%)**
$(x, \_\mathtt{r}, y), (y, \_\mathtt{s}, z) \rightarrow (x, \_\mathtt{t}, z)$	0	**2 (12%)**	**2 (12%)**
$(x, \_\mathtt{r}, y), (x, \_\mathtt{s}, y) \rightarrow (x, \_\mathtt{t}, y)$	0	0	**42 (49%)**
$(x, \mathsf{ype}, \_\mathtt{t}), (x, \mathsf{type}, \_\mathtt{u}) \rightarrow (x, \mathsf{type}, \_\mathtt{v})$	-	-	-
	INDIGO-BM		
Rule pattern	R-GCN	GraIL	INDIGO
$(x, \_\mathtt{r}, y) \rightarrow (x, \_\mathtt{s}, y)$	2 (9%)	**3 (13%)**	**3 (13%)**
$(x, \mathsf{type}, \_\mathtt{t}) \rightarrow (x, \mathsf{type}, \_\mathtt{u})$	89 (20%)	0	**207 (46%)**
$(x, \_\mathtt{r}, y) \rightarrow (y, \_\mathtt{r}, x)$	8 (33%)	0	**14 (58%)**
$(x, \_\mathtt{r}, y), (y, \_\mathtt{s}, z) \rightarrow (x, \_\mathtt{t}, z)$	95 (14%)	130 (19%)	**150 (22%)**
$(x, \_\mathtt{r}, y), (x, \_\mathtt{s}, y) \rightarrow (x, \_\mathtt{t}, y)$	19 (7%)	19 (7%)	**116 (41%)**
$(x, \mathsf{ype}, \_\mathtt{t}), (x, \mathsf{type}, \_\mathtt{u}) \rightarrow (x, \mathsf{type}, \_\mathtt{v})$	5281 (20%)	timeout	**12,288 (47%)**

To understand the kinds of general patterns that our system was able to learn, we have examined its predictions made on v2 of GraIL-BM/FB15K-237 benchmark. We highlight here a few interesting examples of such patterns, which can be naturally expressed as logical rules.

First, INDIGO was able to learn that directors who won an award for a movie were first nominated; for instance, from (*Doctor Zhivago*, awardWinner, *D.Lean*) our system predicted (*D.Lean*, nominatedFor, *Doctor Zhivago*). INDIGO also Learnt transitivity of influencedBy relation; from (*H.Arendt*, influencedBy, *K.Marx*) and (*K.Marx*, influencedBy, *I.Kant*) it predicted (*H.Arendt*, influencedBy, *I.Kant*). Finally, INDIGO Learnt more complex patterns, such as the fact that movies filmed in the official language of a country were produced in that country.

3.6 Summary

In this chapter, we have proposed a novel GNN-based approach to inductive KG completion which significantly outperforms state-of-the-art approaches. The key novelty of our approach is that it encodes KGs using a one-to-one correspondence between triples in the KG and elements of nodes' feature vectors in the graphs processed by the GNN. We have implemented our approach in a system called INDIGO, and showed in our experiments that INDIGO has achieved superior performance compared with the baselines on the existing inductive benchmarks and a new benchmark proposed by us. Moreover, INDIGO also empirically demonstrates greater efficiency in both training and testing, and better capability to capture inference patterns represented using logical rules.

While our work provide valuable insights, there are certain limitations that should be acknowledged. First of all, the size of the annotated graph generated by the encoder can be much larger than the size of graphs generated by R-GCN and CompGCN. Even though we showed in Section 3.3.1 that it is quadratic only when every two constants are connected by a triple in the KG or the candidate set, the practical implications of its size growth in more typical cases should not be overlooked, and it is important to monitor the size growth and even optimise it more efficiently. In addition, the dimension of the feature vectors in our pair-wise encoding rely fully on the number of relations and types. This can potentially lead to a rapid increase in vector dimensions when there are huge number of relations or types in the KGs, which is common in practice for some specific domains such as biomedical KGs. Moreover, in the experiment results, INDIGO demonstrated poor performance in C-Hits@k and C-MRR, as the baselines use entity corruption in negative sampling for training, while our sampling strategy in training does not favour any particular metric. However, it is still worth exploring about designing a negative sampling strategy for training which can result in balanced performance in all the ranking-based metrics. Another limitation of our approach is that all predicted triples involve only the relations and types mentioned in the original KG; there has been recent work on methods that are able to complete the KG with triples over unseen relations and types [45, 106, 121, 172], and it would be interesting to see if our techniques can be extended to this setting. Moreover, our approach is applicable to the case where the unseen constants are isolated and not connected to any of the other constants, as the structural information is needed for making predictions. A possible way to

address the limitation is to combine our approach with the approaches utilising side information such as text description [34, 118, 136, 157, 168].

We see several interesting future directions for this work. First of all, it would be interesting to exploit the logic characterisation for our novel pair-wise encoding approach based on our discussion in Section 3.4. In addition, it is also worth exploring the potential applications of our approach in real-world settings such as KG-enhanced recommendation systems. We will present a work about this in Chapter 4. Moreover, it is interesting to explore how the patterns learnt by our model can be expressed symbolically as logical rules. Finally, as shown in Table 3.12, there are very few high-confidence rules in the existing benchmarks, and we also see it a valuable direction to design inferential benchmarks that can evaluate the systems' ability to capture inference patterns in practice. We will discuss our work on such a benchmarking approach in Chapter 5.

Table 3.13: Variance of results on the benchmarks in %; R, G, H, and I stand for R-GCN, GraIL, the system of Hamaguchi et al., and INDIGO, respectively.

	Bench-mark		Accuracy				AUC				Precision				Recall			
			R	G	H	I	R	G	H	I	R	G	H	I	R	G	H	I
GraIL-BM	FB15K-237	v1	.00	.04	-	.00	.00	.03	-	.00	.00	.17	-	.02	.00	.05	-	.00
		v2	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.01	-	.00
		v3	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.01	.00	.01	-	.00
		v4	.00	.00	-	.01	.00	.01	-	.01	.00	.00	-	.04	.00	.01	-	.00
	NELL-995	v1	.00	.01	-	.05	.00	.01	-	.02	.00	.01	-	.07	.00	.01	-	.00
		v2	.00	.02	-	.00	.00	.01	-	.00	.00	.02	-	.01	.00	.01	-	.00
		v3	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.01	.00	.00	-	.00
		v4	.00	.01	-	.01	.00	.00	-	.01	.00	.00	-	.02	.00	.00	-	.00
	WN18RR	v1	.00	.00	-	.05	.00	.01	-	.05	.00	.00	-	.07	.00	.01	-	.00
		v2	.00	.00	-	.00	.00	.00	-	.01	.00	.00	-	.01	.00	.00	-	.00
		v3	.00	.00	-	.01	.00	.01	-	.00	.00	.00	-	.03	.00	.02	-	.00
		v4	.00	.00	-	.02	.00	.00	-	.02	.00	.00	-	.06	.00	.00	-	.00
Hamaguchi-BM	h-1K		.00	.01	.00	.02	.00	.00	.01	.03	.00	.00	.01	.04	.00	.02	.00	.00
	h-3K		.00	.00	.00	.01	.00	.00	.01	.01	.00	.00	.01	.04	.00	.00	.00	.00
	h-5K		.00	.00	.00	.01	.00	.00	.01	.00	.00	.00	.01	.02	.00	.00	.00	.00
	t-1K		.00	.01	.01	.02	.00	.01	.01	.04	.00	.00	.02	.04	.00	.02	.01	.00
	t-3K		.00	.01	.00	.09	.00	.00	.00	.01	.00	.00	.01	.03	.00	.01	.00	.00
	t-5K		.00	.00	.02	.06	.00	.00	.03	.00	.00	.00	.05	.01	.00	.00	.00	.00
	b-1K		.00	.01	.00	.03	.00	.01	.01	.02	.00	.00	.01	.06	.00	.02	.00	.00
	b-3K		.00	.00	.01	.03	.00	.00	.01	.01	.00	.00	.02	.08	.00	.00	.00	.00
	b-5K		.00	.00	.01	.03	.00	.00	.01	.00	.00	.00	.01	.08	.00	.00	.00	.00
INDIGO-BM			.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00

	Bench-mark		C-Hits@1				C-Hits@3				C-Hits@10				C-MRR			
			R	G	H	I	R	G	H	I	R	G	H	I	R	G	H	I
GraIL-BM	FB15K-237	v1	.00	.01	-	.02	.00	.00	-	.02	.00	.00	-	.01	.00	.00	-	.01
		v2	.00	.01	-	.05	.00	.00	-	.03	.00	.00	-	.03	.00	.00	-	.02
		v3	.00	.01	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00
		v4	.00	.01	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.01
	NELL-995	v1	.00	.01	-	.00	.00	.01	-	.00	.00	.04	-	.00	.00	.01	-	.00
		v2	.00	.01	-	.02	.00	.00	-	.01	.00	.00	-	.01	.00	.00	-	.01
		v3	.00	.00	-	.00	.00	.01	-	.00	.00	.01	-	.00	.00	.00	-	.00
		v4	.00	.01	-	.00	.00	.00	-	.01	.00	.00	-	.00	.00	.00	-	.00
	WN18RR	v1	.00	.01	-	.03	.00	.00	-	.02	.00	.00	-	.01	.00	.00	-	.01
		v2	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00
		v3	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00
		v4	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00	.00	.00	-	.00
Hamaguchi-BM	h-1K		.00	.00	.03	.02	.00	.00	.02	.01	.00	.00	.01	.00	.00	.00	.01	.00
	h-3K		.00	.00	.01	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
	h-5K		.00	.00	.00	.01	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
	t-1K		.00	.00	.02	.00	.00	.00	.01	.00	.00	.00	.01	.00	.00	.00	.01	.00
	t-3K		.00	.00	.00	.01	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
	t-5K		.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01
	b-1K		.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
	b-3K		.00	.00	.00	.02	.00	.00	.00	.01	.00	.00	.00	.01	.00	.00	.00	.01
	b-5K		.00	.00	.01	.00	.00	.00	.00	.00	.00	.00	.01	.01	.00	.00	.00	.00
INDIGO-BM			.00	.00	.00	.01	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00

Chapter 4

InKER: GNN-Based Inductive Knowledge-Enhanced Recommender Systems

4.1 Introduction

Recommender systems [3] encompass a range of techniques aimed at suggesting to users relevant items, such as videos or online products, based on past interactions (*collaborative filtering systems*), attributes associated to items and users (*content-based systems*), or user constraints and external knowledge (*knowledge-based systems*). Such systems typically suggest items by predicting a rating for a given user and item, or by identifying a set of top items for a given user.

The interactions between m users and n items can be represented as an incomplete $m \times n$ *ratings matrix* [22, 61], where the (u, i) component contains the rating given by user u for item i (if this rating is known) or it remains blank (if no such rating is available). Ratings are usually numeric, often discrete values (e.g., a 1–5 stars rating, or a 0–1 binary rating corresponding to 'likes' and 'dislikes'), and this setting is often referred to as recommendation with *explicit feedback*. There are situations, however, where there is no mechanism for users to specify a negative response (e.g., only a 'like' button is available) or when there is only information about customer actions (e.g., clicks or purchases); in this setting, referred to as recommendation with *implicit feedback*, ratings are *unary*: each matrix entry either contains a 1 or no value, where the latter is not an indication of the user disliking the item.

The goal of recommender systems is then to complete missing entries in the (typically sparse) matrix. Thus, recommendation with explicit and implicit feedback can be seen as ML regression and classification problems, respectively.

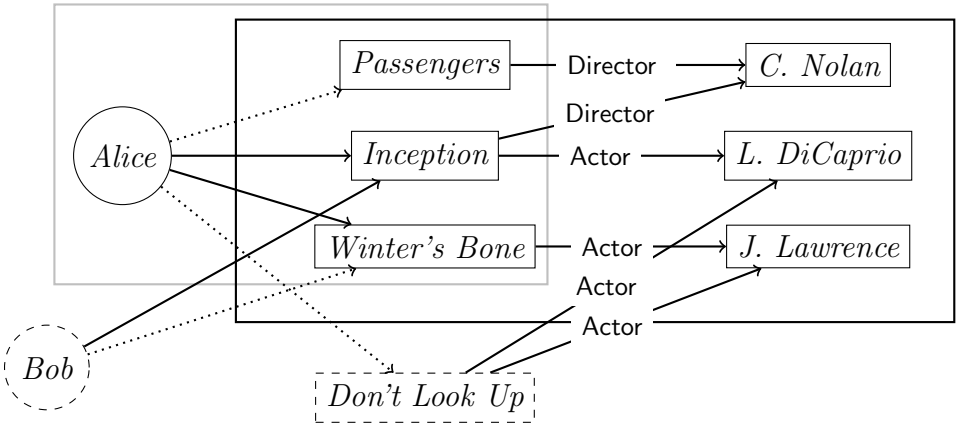


Figure 4.1: Example of inductive knowledge-enhanced implicit-feedback recommendation. Solid arrows in the grey and the black frames contain a bipartite graph of user-item interactions and a KG, respectively, which are both available for training; the user and the item outside the frames (i.e., *Bob* and *Don't Look Up*) and their solid arrows are the new information made available only after training; dotted arrows are interactions to predict.

Matrix-Factorisation-based approaches are one of the prominent types of approaches for rating prediction [48, 69, 85, 113], which achieves remarkable performance when sufficient historical interaction data is provided. However, one key limitation of these approaches is that they perform poorly when the matrix is sparse. Furthermore, they are inherently *transductive*: training and test data are tightly integrated in the matrix, in the sense that they mention the same sets of users and items; as a result, models cannot easily predict ratings for 'out-of-sample' users and items, which were not mentioned in the training set.

A common way of addressing the sparsity problem is to use additional information about items. In recent years, there has been increasing interest in exploiting external KGs to improve the accuracy, diversity, and interpretability of recommendation results [139, 142, 174]. A key limitation of existing models, however, is that they are also restricted to transductive settings. This is illustrated in Figure 4.1, where a system trained on the interaction graph within the grey frame and the KG in the black frame can predict the interaction (*Alice*, *Passengers*), but cannot predict the interactions (*Bob*, *Winter's Bone*) and (*Alice*, *Don't Look Up*) without retraining the model.

Recommendation methods in *inductive settings*, where predictions are made also for objects unseen during training, have been recently considered; however, state-

of-the-art inductive recommender systems, such as IDCF [155] and IGMC [177], do not support KGs. These systems also present additional limitations: IDCF is only partially inductive (although it can make predictions for unseen users, it requires retraining for unseen items), whereas IGMC suffers from scalability issue for inputs with large numbers of users and items.

Moreover, there are some inherent correlations between KG completion and recommender systems. In particular, KG completion aims at predicting missing triples, which can be seen as links between constants. Similarly, recommender systems aim at predicting user-item interactions, which can also be seen as missing links between users and items. Hence, the essential problems for both tasks are link prediction, which aims to predict missing links between two objects. This allows us to extend the methodology for KG completion to recommender systems.

In this chapter we propose a novel approach that is effective and scalable in the inductive setting, and which can seamlessly incorporate KGs to enhance the quality of recommendations. Our approach builds upon the neural architecture for inductive KG completion introduced in Chapter 3, and can be conceptually divided into three stages. In the first stage, user-item ratings and external KGs are jointly encoded as a graph with nodes annotated by numeric feature vectors in a transparent and direct way. In the second stage, the aforementioned graph is fed as input to a GNN. Finally, in the third stage, the recommendation results are decoded directly from the output layer of the GNN.

Although GNNs are increasingly being used as components of recommender systems [139, 142, 155, 177], ours is, to the best of our knowledge, the first approach that exploits the unique features of GNNs to address the inductive recommendation problem in its full generality while at the same time incorporating KGs. Finally, in contrast to systems such as RippleNet [139] and KGCN [142], our approach is able to produce ratings for both implicit- and explicit-feedback cases, which makes it suitable for a wide range of applications.

In this work, we have implemented our approach in a system called InKER (Inductive Knowledge-Enhanced Recommendation), and compared its performance with that of state-of-the-art baselines on several recommendation benchmarks involving KGs. Our results show that our system, while being more general than the baselines, is able to outperform them on the restricted cases to which the baselines are applicable. Furthermore, our system displays robust and scalable performance across different scenarios. Finally, we also analysed the impact on system's performance of

using different types of GNNs, as well as the significance of incorporating KGs in situations where the rating matrix is sparse.

The rest of this chapter is organised as follows. In Section 4.2, we will first review the related literature about approaches for GNN-based recommendation and knowledge-enhanced recommendation. We will then present some necessary preliminaries in Section 4.3, and introduce our approach in Section 4.4. Finally, we will show an empirical study about the evaluation of our approach and related baselines in Section 4.5, and conclude this chapter with a summary in Section 4.6.

4.2 Related Work

Our work is closely related to both GNN-based recommender systems and knowledge-enhanced recommender systems. In this section, we discuss the related work about each of the two topics.

4.2.1 GNN-Based Recommender Systems

Recent years have witnessed an increasing interest in the use of GNNs on the recommender systems. These approaches are mainly divided into three categories based on the type of information they utilise: user-item interactions only; user-item interactions and KGs; and user-item interactions and social networks.

In general, the approaches using only user-item bipartite graphs [54, 75, 131, 160, 170, 171, 175, 177] learn the representations of users and items by aggregating the information of their neighbours. For example, GC-MC [131] applies a GCN-based encoder on the bipartite graph to learn the embeddings of users and items, and a bilinear decoder to estimate the probability distribution over different ratings and reconstruct the bipartite graph with ratings. A drawback of GC-MC is that it requires operating on the full graph Laplacian during training, which is not efficient for large-scale recommendation data. PinSage [170] is then proposed to alleviate the problem by constructing localised convolutions via random walks. This work also designed strategies such as minibatching and MapReduce for effective GPU training and inferring. As discussed in Section 2.5, one of the inherent limitations of general GNN, as well as most of the GNN-based systems, is the over-smoothing problem when stacking many GNN layers [75]. This problem is addressed in STAR-GCN [175], which builds a stack of encoder-decoder blocks. Instead of stacking multiple convolutional layers, it stacks multiple identical blocks of encode-decoder structure, such as the structure

in GC-MC. GC-MC, PinSage, and STAR-GCN can all be *inductive* when side information is provided. In particular, they can initialise the vectors of users and items with textual and visual features, which are, however, not always available. Instead of utilising side information, IGMC [177] first extracts the a one-hop subgraph for each user-item pair, then assigns labels to the nodes in the subgraph based on their distance to the aforementioned pair, and finally predicts the rating for a use-item pair by applying a GNN on its subgraph. This framework is fully inductive, but is restricted by its scalability: when predicting ratings for millions of movies, IGMC needs to extract millions of subgraphs, which is very time-consuming and not practical. Moreover, LightGCN [54] further simplifies the convolution operations for the message passing among users and items. Similar to the MF approaches discussed in Section 4.1, GNN-based approaches for recommender systems with only the user-item bipartite graph are not able to maintain a robust performance when the rating matrix becomes sparse. Therefore, there is rich literature exploiting external information, such as KGs and social networks, to address the sparsity problem.

Our work is more related to the GNN-based approaches for recommender systems supporting KGs [139, 142, 140, 147, 148], which serve as useful resources to provide auxiliary knowledge when there is not adequate interaction information. RippleNet [139] is an early system utilising GNN to learn the feature vectors of users. It first collects a set of items (named *relevant entities*) that the user has interacted with. Then, it propagates information among a k-hop set of entities starting from the relevant entities, which is called *ripple set*, to update the vectors of users. In this way, RippleNet can discover users' potential interests by aggregating their historical preferences on the entities in the KG. Different from RippleNet, KGCN [142] aims to learn the feature vectors of items by aggregating the vectors of their neighbours in the KG. To make the system more efficient and scalable, it only samples a fixed size of neighbours. Following this work, KGCN-LS [140] further devised a label-smoothness regularisation module to learn the importance scores of different edges in the KG. The aforementioned systems only apply GNNs on the KG, whereas KGAT [147] further extends the message passing to the user-item bipartite graph. It first constructs a *collaborative KG* by merging the user-item bipartite graph and the KG. Then a TransR model [77] is employed to initialise the representations of the entities in the graph, which are further fed into a GNN to compute the representations of items and users. In addition, an attention mechanism [134] is applied to assign different importance scores to different neighbours, which is of benefit to the quality of the representations and also the explainability of the recommendation results. KGIN [148] is a recently

proposed KG-enhanced recommendation approach, which explores the intents behind a user-item interaction model each intent as an attentive combination of KG relations. However, a common limitation of these approaches is that they are all *transductive*. In particular, they are not applicable to the case where new users or items are added during testing and require retraining, which poses a significant challenge in dynamic and evolving recommendation scenarios.

There are a number of GNN-based approaches incorporating the social networks as side information [42, 153, 154], where nodes represent users and edges represent social relationships between users. Social recommendation can improve the explainability of the systems by showing reasons for predictions such as 'you might be interested in this item since your friends bought it'. For instance, DiffNet [153] first employs an embedding layer and a fusion layer to initialise the vectors of user and item with different input features. A GNN is then used to aggregate information along the social links. Finally, a prediction layer is adopted to compute the probability of potential interactions based on the updated vectors. A drawback of DiffNet is that it assumes that each friend has equal social influence on the user, which is not that reasonable since users are more likely to be influenced by friends with stronger social connections. Furthermore, it only applies GNN on the social networks, and the item vectors are static. In order to address these limitations, GraphRec [42] is proposed to apply different GNNs to update the vectors for users and items separately. In addition, it also employs an attention mechanism to differentiate the social influence among different friends. Similarly, DANSER [154] utilises two dual GATs to model social homophily of users, social influence of users, item-to-item homophily, and item-to-item influence. In this way, the graph structure of both user-item graphs and social networks can be fully exploited.

4.2.2 Knowledge-Enhanced Recommender Systems

In addition to GNN-based approaches, it is also becoming popular to devise approaches that are not GNN-based for knowledge-enhanced recommender systems [101, 141, 165, 167]. For instance, MKR [141], which is evaluated in our work, employs a multi-task learning approach by utilising the KG embedding task to improve the performance of the recommendation task. In the recommendation module, they use a MLP to extract the user embeddings, and the cross & compress units to extract the item embeddings. The prediction score of a user-item pair is computed by the inner product of the user and item embeddings. Palumbo et al. [101] apply node2vec [47], a graph representation algorithm, to learn the embedding of users and items. The

prediction score is then computed by the cosine distance of the user and item embeddings. ACAM [165] takes the information in the KG as item attributes, and the items that the users have interacted with as user attributes. Then, it employs a co-attention mechanism to capture the associations among different attributes of users or items. Finally, an MLP is utilised to compute the scores for recommendation. KGCL [167] proposes a KG augmentation schema to diminish the noise in the KG, and also utilises supplementary supervision signals from the KG augmentation process to steer a cross-view contrastive learning framework. Although the approaches discussed above achieved remarkable performance in the *transductive* setting, they are not compatible with the *inductive* setting.

A different family of KG-enhanced recommendation systems exploit rule learning approaches to extract a set of rules from the KG and exploit these rules to guide the training of a model [83, 84]. For example, RGRec [83] consists of a rule learning module to extract high-quality rules, and a GNN-based recommendation module guided by the extracted rules. One drawback of these approaches is that the number and type of learnt rules are very limited.

4.2.3 Discussion

In this section, we have reviewed the ideas and prominent works for GNN-based recommender systems and knowledge-enhanced recommender systems. It is also worth mentioning that, as discussed in Section 3.2.3, GNNs have also been applied to the KG completion problem, and systems such as R-GCN [115], GraIL [127], and our INDIGO system proposed in Chapter 3 has proved effective in inductive KG completion scenarios. GC-MC is an extension of R-GCN to the recommendation problem, while GraIL and IGMC share a similar idea of extracting an enclosing subgraph for the target pair. Although these problems are conceptually related and some ideas can be useful in both settings [24], there are several important differences. On the one hand, in recommendation problems, there is a clear separation between users, items, and other KG constants, as well as between the interaction graph and the KG. On the other hand, the recommendation problem involves a rating domain, and scenarios involving recommendations with explicit feedback have no analogue in the setting of KG completion.

In conclusion, existing recommender systems supporting KGs, either GNN-based or not, have been designed for transductive settings, where predictions are only made for users and items observed during training; furthermore, existing inductive systems, which are able to make predictions involving unseen items and users do not support

KGs. To the best of our knowledge, there has been very little literature discussing the inductive recommender systems supporting KGs. In the rest of this chapter, we will introduce our approach to inductive KG-enhanced recommendation, which builds on the KG completion architecture implemented in INDIGO, and specifically on the idea of a pairwise encoder elaborated in Section 3.3.1. We will show that our approach is effective in inductive settings, and can seamlessly incorporate KGs to enhance the quality of recommendations.

4.3 Inductive KG-Enhanced Recommendation

In this work, we consider two types of user-item interaction: with implicit feedback (i.e., with unary ratings) and explicit feedback (i.e., with numeric ratings). We model a set of interactions with implicit feedback as a bipartite *interaction graph* $\mathcal{B} = (\mathcal{U}, \mathcal{I}, Q)$, with $\mathcal{U}$ and $\mathcal{I}$ disjoint sets of *users* and *items*, respectively, and $Q \subseteq \mathcal{U} \times \mathcal{I}$ the set of edges; here, an edge (u, i) in Q represents the fact that user u has *interacted* with (e.g., clicked on, purchased, or liked) item i. For interactions with *explicit feedback*, the graph is a tuple $(\mathcal{U}, \mathcal{I}, Q, \mathtt{V})$, where $\mathcal{U}, \mathcal{I}, Q$ are as before, and $\mathtt{V}$ is a *rating function* labelling edges in Q with values from a numeric domain; intuitively, $\mathtt{V}(u, i)$ indicates the rating that user u gives to item i. The *rating domain* of the graph is the range of $\mathtt{V}$; for example, it may be consecutive integers $\{k_{\min}, \ldots, k_{\max}\}$, the natural numbers $\mathbb{N}$, or a continuous interval $[k_{\min}, k_{\max}]$. In practice, the rating domain is usually known in advance.

We exploit side information in the form of KGs to enhance the quality of recommendations. In what follows, for $\mathcal{K}$ a KG used with $\mathcal{B}$, we assume that $\mathcal{I} \subseteq \mathsf{Consts}(\mathcal{K})$—that is, all items $\mathcal{I}$ in an interaction graph $\mathcal{B} = (\mathcal{U}, \mathcal{I}, Q)$ (or $\mathcal{B} = (\mathcal{U}, \mathcal{I}, Q, \mathtt{V})$) are constants in the KG. In practice, $\mathcal{I}$ and $\mathsf{Consts}(\mathcal{K})$ are often disjoint, and an item-constant *alignment* (i.e., a function from $\mathcal{I}$ to $\mathsf{Consts}(\mathcal{K})$) is used to bridge the gap between them; however, this technicality is inessential to our work, so we abstract it away by assuming that the alignment is the identity.

Roughly speaking, the goal of *knowledge-enhanced recommendation* is to predict, given an interaction graph $\mathcal{B}$ and a KG $\mathcal{K}$, the interaction of a candidate user-item pair; here, it is assumed that the user and the item are mentioned in $\mathcal{B}$ separately, but not as a pair (i.e., the interaction for this pair is not yet known). In the implicit-feedback case, the prediction for a pair (u, i) is a Boolean value, showing whether u would interact with i or not, and in the explicit-feedback case it is a value from the rating domain of $\mathcal{B}$, showing a likely rating of the pair.

To formalise knowledge-enhanced recommendation as an ML problem, we should specify what is fixed for both training and testing, and what is the input to the ML model. Following Aggarwal et al. [3], in the *transductive setting* everything is fixed except the user-item interactions (i.e., the users, the items, and the contents of the KG are the same for training and testing).

The *inductive setting* is more general. In its full generality, only the type of interactions, the rating domain (when applicable), and the set of relations $\mathcal{R}$ of the KGs are fixed, whereas everything else constitutes the input. Formally, the goal is to learn, given an interaction type equipped with a rating domain (if applicable) and a set $\mathcal{R}$ of relations, a function $f(\cdot, \cdot, \cdot)$ that takes as arguments an interaction graph of the relevant type, the KG over relations $\mathcal{R}$, and a candidate user-item pair, and returns the pair's predicted value.

There are, however, several intermediate cases. In this work we consider two of them, which we anticipate to be the most relevant in practical scenarios where the KG is used to describe items (but not users):

- *inductive knowledge-enhanced recommendation with known users*, which is the same as the fully inductive case except that the set of users is fixed (i.e., the same for training and testing), and

- *inductive knowledge-enhanced recommendation with known users and KG*, which is the same except that both users and the KG are fixed (i.e., only the items in the interaction graphs for training and testing may vary).

Observe that the latter setting is close to the transductive setting: since the KG is the same for training and testing, and all items are constants in the KG, all 'unseen' items are known to the system as constants. This observation enables the application of KG-aware transductive systems, such as RippleNet and KGCN, to this setting; this is something that we will exploit later on in our experiments.

4.4 Approach to Recommendation

Our approach to recommendation is inspired by the neural architecture for inductive KG completion proposed in Chapter 3, which we have adapted and extended to address the inductive KG-enhanced recommendation problem.

4.4.1 Overview

Similar to the pipeline proposed in Section 3.3.1, we realise a recommendation function $f(\cdot, \cdot, \cdot)$ using three main components:

- an *encoder*, which transforms an input interaction graph $\mathcal{B}$, a KG $\mathcal{K}$, and candidate user-item pairs Λ into a graph G with numeric vector annotations;
- a *GNN*, which updates the annotations of nodes in G throughout multiple layers on the basis of their neighbours' annotations;
- a *decoder*, which extracts the predictions $f(\mathcal{B}, \mathcal{K}, \lambda)$ for each $\lambda \in \Lambda$ from the output layer of the GNN.

As explained in Section 4.3, we tacitly assume that items in $\mathcal{B}$ are constants of $\mathcal{K}$, while the elements of each candidate user-item pair in Λ are assumed to occur in $\mathcal{B}$ separately, but not as a pair. For uniformity, we will collectively refer to users and constants (including items) as *objects*.

The main novelty of our approach lies in the design of the encoder and the decoder. First, our encoder generates a single graph for a set of candidate user-item pairs, which is important for scalability; this is in contrast to other approaches such as IGMC [177], which generate a different graph for each candidate pair.

Second, existing GNN-based recommendation systems encode objects as nodes of the input graph to the GNN, and reserve the initialisation of the input feature vectors to encode additional information on these objects; when such information is unavailable (which is commonly the case), feature vectors are initialised at random [139, 142]. In contrast, in our approach, each node of the input graph to the GNN represents a *pair* of objects, which allows us to explicitly encode the relevant *structural* information into the initial feature vectors of the nodes.

Finally, most existing GNN-based recommender systems use GNNs to learn the embeddings of users and items separately, and rely on an ad-hoc output function (e.g., inner product) to make predictions for pairs. In contrast, our system decodes the predictions directly from dedicated elements of the feature vectors at the outermost layer of the GNN.

For the input of the GNN, we consider a δ-annotated graph G and an enumeration id, which are defined in Section 3.3, and we assume the types mentioned in Section 3.3 are omitted in the remainder of this section. Particularly, in this chapter, we take $\delta = 2{\cdot}|\mathsf{Rels}(\mathcal{K})|{+}1$, so that the i-th element of each feature vector for $i = 2{\cdot}|\mathsf{Rels}(\mathcal{K})|{+}1$ is used to encode recommendations.

In the remainder of the section, we will present the technical details of encoding in Section 4.4.2, GNN aggregation in Section 4.4.3, and decoding in Section 4.4.4.

4.4.2 Encoding

In this section, we propose an encoder that transforms an interaction graph $\mathcal{B}$, a KG $\mathcal{K}$, and candidate pairs Λ into an δ-annotated graph G. Our encoding captures the connectivity of $\mathcal{B}$ and $\mathcal{K}$ by means of encoding their structure explicitly in the initial feature vectors of G.

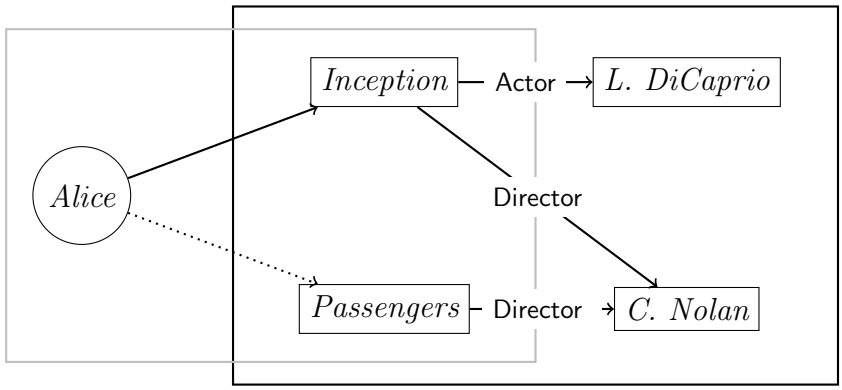
The literature on KG-enhanced recommender systems suggests two strategies for encoding $\mathcal{B}$ and $\mathcal{K}$ into the input of a GNN. The first is to encode one or both of them separately and apply a dedicated GNN to each of the encodings [142]. However, the limitation of this strategy is that these GNNs do not directly exchange information during their execution, which limits the effectiveness of KGs in providing information about associations among objects. To address such limitation, the second strategy, which we adopt in this work, is to apply a GNN to a unified graph encoding both $\mathcal{B}$ and $\mathcal{K}$ [107, 147]. Please note that, sometimes the interaction graph $\mathcal{B}$ is also seen as triples so that $\mathcal{B}$ and $\mathcal{K}$ are not distinguished from each other. However, we highlight the difference between $\mathcal{B}$ and $\mathcal{K}$ so that our approach is able to deal with the explicit cases.

The encoding of this work builds up on the idea of pair-wise encoding elaborated in Section 3.3.1. However, there are some technical differences between the encoding approaches of these two works, which we highlight here. First of all, in addition to $\mathcal{K}$ and Λ, in this work we also take the interactions in $\mathcal{B}$ into our consideration, and thus each node of the annotated graph corresponds to a pair of objects mentioned together in an edge of $\mathcal{B}$, $\mathcal{K}$, or Λ. Second, in the encoding for KG completion, we only add a node $v_{c,d}$ to graph G if both c and d are constants and $c \preceq d$. However, in this work, we aim to also cover the case of user-item interactions. Hence, a node $v_{c,d}$ can also be added to graph G if c is a user and d is an item. Last but not least, in the feature vectors of each node $v_{c,d}$ in G, each element can represent not only the existence of a corresponding relation or its reverse as defined in the completion work, but also the existence of the feedback. The value for each element representing the feedback depends on the type of interaction. In particular, in the implicit-feedback case, it is either 1 when (c, d) is an edge in $\mathcal{B}$ (i.e., when we know about the interaction), and 0 when $(c, d) \in \Lambda$ (i.e., we would like to check whether the interaction is likely); in the explicit-feedback case, the element is the rating of (c, d) in $\mathcal{B}$, if the pair is in $\mathcal{B}$, or 0, if the pair is in Λ.

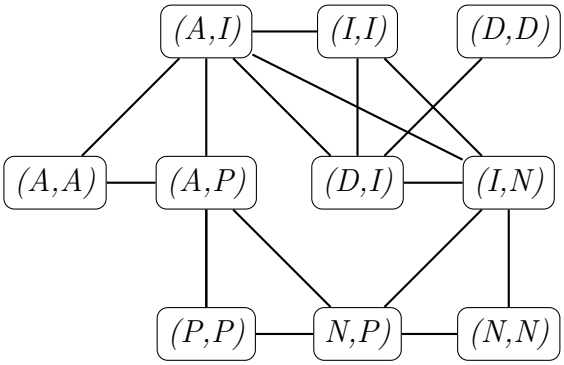
Definition 5. *The* encoding *of an interaction graph $\mathcal{B}$, KG $\mathcal{K}$, and candidate pairs Λ is the δ-annotated graph G where*

- *G has a node $v_{c,c}$ for each object c in $\mathcal{B}$ and $\mathcal{K}$, and a node $v_{c,d}$ for each pair (c, d) of objects such that either $(c, d) \in \mathcal{B} \cup \Lambda$, or $c \prec d$ and $\mathcal{K}$ contains a triple (c, r, d) or (d, r, c) for some $r \in \mathsf{Rels}(\mathcal{K})$;*
- *G has an edge between different nodes v_X and v_Y if pairs X and Y have an object in common;*
- *the feature vectors of nodes in G are defined as*
  - *$(\mathbf{v}_{c,d})_{\mathsf{id}(r)} = 1$ for all $(c, r, d) \in \mathcal{K}$ with $c \preceq d$,*
  - *$(\mathbf{v}_{c,d})_{\mathsf{id}(r^-)} = 1$ for all $(d, r, c) \in \mathcal{K}$ with $c \preceq d$,*
  - *$(\mathbf{v}_{c,d})_{\delta} = y_{c,d}$ for all edges (c, d) in $\mathcal{B}$, where $y_{c,d}$ is 1 in the implicit-feedback case and the rating of (c, d) in $\mathcal{B}$ in the explicit-feedback case,*
  - *all other elements are 0.*

An example of the encoding is given in Figure 4.2. The input graph combining the interaction and knowledge graphs is given in Figure 4.2(a), where *Alice* and *Passengers* constitute the only candidate user-item pair in Λ since we aim to find out whether Alice would like to watch that movie. The annotated graph resulting from applying the encoder is summarised in Figure 4.2(b). We can observe that only pairs of objects that are directly connected in the input graph have a corresponding node in the annotated graph; for instance, there is no node (A, N) because there is no triple connecting *Alice* and *C. Nolan* in the input. Finally, Figure 4.2(c) provides the feature vector for the node (A, I) in the encoded graph and indicates the correspondence between each of its components and a KG triple (or interaction pair) relating those objects; for instance, the component corresponding to the pair $(\mathit{Alice}, \mathit{Inception})$ is 1 because this interaction appears in the input, whereas the component for triple $(\mathit{Alice}, \mathsf{director}, \mathit{Inception})$ is 0 because this triple does not occur in the KG.



(a)



(b)

$$
\begin{aligned}
\textit{(A,I)} &: [0,0,0,0,1] \\
(\textit{Alice}, \mathsf{Actor}, \textit{Inception}) &: 0 \\
(\textit{Alice}, \mathsf{Director}, \textit{Inception}) &: 0 \\
(\textit{Inception}, \mathsf{Actor}, \textit{Alice}) &: 0 \\
(\textit{Inception}, \mathsf{Director}, \textit{Alice}) &: 0 \\
(\textit{Alice}, \textit{Inception}) &: 1
\end{aligned}
$$

(c)

Figure 4.2: (a) Example of an interaction graph (grey frame) and KG (black frame); (b) the graph part of their encoding; (c) the feature vector of node *(A,I)* with justifications, which build an one-to-one correspondence between the existence of a fact with an element in the vector

Similar to 3.3.1, we conclude this section with a brief discussion on the size of the annotated graph generated by the encoder. The difference here is that we also

consider the number of users, items, and interactions in the interaction graph. Let $\mathcal{B}$, $\mathcal{U}$, Q, $\mathcal{K}$, $\mathsf{Consts}(\mathcal{K})$, Λ be as before. By construction, the δ-annotated graph has up to $|\mathcal{U}|+|\mathsf{Consts}(\mathcal{K})|+|Q|+|\mathcal{K}|+|\Lambda|$ nodes and up to $d*(|\mathcal{U}|+|\mathsf{Consts}(\mathcal{K})|+|Q|+|\mathcal{K}|+|\Lambda|)$ edges, with d the average degree of an object in P, Λ, or $\mathcal{K}$. In theory, d is bounded by $|\mathcal{U}| + |\mathsf{Consts}(\mathcal{K})|$, and hence the size of the encoding can be quadratic in the size of the input. This bound, however, is only approached when most pairs of objects in $\mathcal{U} \cup \mathsf{Consts}(\mathcal{K})$ are connected by an edge in $\mathcal{B}$, $\mathcal{K}$, or Λ, which hardly happens in practice since these graphs are typically sparse. So, d is typically small and the encoding size is essentially linear in practice.

4.4.3 GNN Aggregation

Analogous to the completion approach proposed in Chapter 3, in this work, we also rely on a GNN to propagate feature information through the edges of the δ-annotated graph generated by the encoder. The technical details of GNNs are introduced in Section 2.5.

In particular, we adopt GCNs [67] as the GNN structure in our implementation, and take ReLU for the non-linear activation functions in the hidden layers and logistic sigmoid in the output layer. Similar to Section 3.5.5, we also analyse the impact on the system performance of using different GNN variants including GCNs, GATs, and GINs, and the results in Section 4.5.6 show that the systems with GAT and GIN variants are generally inferior to those of GCN-based.

4.4.4 Decoder

The decoder extracts the prediction for each of the candidate pairs Λ from the output graph of the GNN.

By construction, the graph has a node $v_{u,i}$ for each $(u, i) \in \Lambda$; moreover, for $\delta = 2 \cdot |\mathsf{Rels}(\mathcal{K})| + 1$, the vector $(\mathbf{v}_{u,i})^L$ of this node has a dedicated element $(\mathbf{v}_{u,i})^L_\delta$ for the rating, whose value is in $(0, 1)$. Our extraction of the prediction is slightly different from the decoding approach in Section 3.3.3, where our approach depend on the type of interaction we consider, as well as on the associated rating domain (in case of interaction with explicit feedback).

Definition 6. *Let* $\mathsf{Dec} : (0, 1) \to \mathbf{D}$ *be the* decoder *with* $\mathbf{D} = \{\mathsf{true}, \mathsf{false}\}$ *for implicit feedback and the rating domain for explicit feedback. Then, the* predicted value *for a user-item pair* (u, i) *in an* δ*-annotated graph* G *with a node* $v_{u,i}$ *is* $\mathsf{Dec}((\mathbf{v}_{u,i})_\delta)$.

In the implicit-feedback case we can take $\mathsf{Dec}(x) = \mathsf{true}$ if and only if $x \geq \theta$, with $\theta = 0.5$ a dedicated threshold. Some performance metrics on implicit-feedback prediction require a $(0, 1)$-confidence rather than true-false predictions; for these metrics, we use $(\mathbf{v}_{u,i})_\delta$ itself instead of $\mathsf{Dec}((\mathbf{v}_{u,i})_\delta)$ in our experiments. In the explicit-feedback case, we can take $\mathsf{Dec}(x) = \lfloor x \cdot (k_{\max} - k_{\min}) + k_{\min} \rceil$ when $\mathbf{D} = \{k_{\min}, \ldots, k_{\max}\}$ (where $\lfloor \cdot \rceil$ is the nearest integer function), and similar for other $\mathbf{D}$.

Overall, our entire pipeline assumes a trained GNN $\mathfrak{N}$; it takes an interaction graph $\mathcal{B}$, a KG $\mathcal{K}$, and candidate pairs Λ as input, and returns the predicted value $\mathsf{Dec}((\mathbf{v}_{u,i})_\delta)$ for each pair (u, i) in Λ as output, where $v_{u,i}$ is taken from the result of $\mathfrak{N}$ applied to the encoding of $\mathcal{B}$, $\mathcal{K}$, and Λ.

4.5 Evaluation

We have implemented our approach using Python and PyTorch v1.4.0 in a system called InKER (Inductive Knowledge-Enhanced Recommendation). To ensure a fair comparison with baselines that do not support KGs, we also implemented a variant InKER$^-$ of InKER, which ignores the input KG (i.e., considers it to be empty). Experiments were performed on an Intel(R) Xeon(R) machine with 8 cores and a 2.6 GHz CPU 257 equipped with 540 GB of RAM running Fedora 33 (x86_64).

In this section, we will show a comprehensive evaluation of our systems and related baselines on the benchmarks. The rest of this section is structured as follows. Section 4.5.1 and Section 4.5.2 introduces the baselines and benchmarks used in the experiments. Section 4.5.3 presents the technical details of training for our system as well as the compared baselines, and Section 4.5.4 describes the evaluation metrics we adopted in the experiments. The results for the majority of the experiments are discussed in Section 4.5.5. Finally, Section 4.5.6 presents an ablation study and Section 4.5.7 shows a comparison of InKER with IGMC by varying the sparsity of the training and validation data.

4.5.1 Baselines

As far as we know, no recommender system supports the inductive setting in its full generality while at the same time exploiting KGs. So, we used the following systems, which have one of these two capabilities, as baselines. RippleNet [139], KGCN [142], KGNN-LS [140], MKR [141], and RGRec [83] are transductive systems that can take KGs into account; they support only implicit-feedback interactions. All these systems except MKR are GNN-based. GC-MC [131] and IGMC [177] support

inductive settings, but do not take KGs into account. They see recommendation as a multi-classification problem and so support only explicit feedback with a finite rating domain; they can be adapted to the implicit-feedback setting, but cannot deal with unbounded domains such as $\mathbb{N}$. We also used GraIL [127], an inductive KG completion system, as a baseline for the implicit-feedback setting, by treating interaction as a specific relation.

Table 4.1 summarised the compatibility of the baselines and InKER from the following perspectives: if it is inductive, if it supports KG, if it supports implicit feedback, and if it supports explicit feedback. Please note that for GCMC and IGMC, they see recommendation as a multi-classification problem and thus support only explicit feedback with a finite rating domain.

Table 4.1: Compatibility of systems used in the experiments

	inductive	KG	implicit	explicit
RippleNet [139]	no	yes	yes	no
KGCN [142]	no	yes	yes	no
KGNN-LS[140]	no	yes	yes	no
MKR [141]	no	yes	yes	no
RGRec [83]	no	yes	yes	no
GCMC [131]	yes	no	yes	yes
IGMC [177]	yes	no	yes	yes
GraIL [127]	yes	yes	yes	no
InKER	yes	yes	yes	yes

4.5.2 Benchmarks

We compared InKER$^{(-)}$ with all the applicable baselines on four (or two) inductive modifications of each of several existing benchmarks from two groups [142, 147] for transductive KG-enhanced recommendation. In particular, we have a modification for a combination of $X \in \{\text{Im, Ex}\}$ and $Y \in \{\text{'fix.', 'var.'}\}$; here X values indicate the implicit- and explicit-feedback versions, while Y values indicate the settings of known (i.e., fixed) users and KG, and known users and varying KG, as discussed above.

Our benchmarks are the inductive versions of the first group of the transductive benchmarks [142]. They are based on the implicit- and explicit-feedback versions of three publicly available sets of user-item interactions: Book-Crossing (with books as items), MovieLens-1M (with movies as items), and Last.FM (with artists as items), and relevant fragments $\mathcal{K}_{\text{BC}}$ for Book-Crossing, $\mathcal{K}_{\text{ML}}$ for MovieLens-1M, and $\mathcal{K}_{\text{LF}}$ for Last.FM, of the Microsoft Satori KG, which is an KG internally used

by Microsoft. We name these benchmarks as BC-X_Y, ML-X_Y, and LF-X_Y for Book-Crossing, MovieLens-1M, and Last.FM respectively. Each interaction in each BC-E_Y (i.e., the explicit-feedback inductive versions of Book-Crossing) is additionally labelled by a 0-to-10-star rating. The rating domain of each ML-E_Y is 1 to 5 stars. In turn, the rating domain of each LF-E_Y is $\mathbb{N}$; here, a 'rating' is the number of times a record of the artist has been listened to by the user on the last.fm website.

Each AB-Im_Y is the inductive version of the benchmark in the second group [147]. They are based on the implicit interactions from Amazon-Books and a relevant fragment $\mathcal{K}_{\mathrm{AB}}$ of Freebase KG. As required in our formalisation, all items of the interactions have corresponding entities in its KG ($\mathcal{K}_{\mathrm{BC}}$, $\mathcal{K}_{\mathrm{ML}}$, $\mathcal{K}_{\mathrm{LF}}$, or $\mathcal{K}_{\mathrm{AB}}$); as explained, this correspondence is established by an alignment provided as a part of the benchmark.

Table 4.2 shows the statistics of the graphs.

As usual in the field [142], our benchmarks do not explicitly provide interaction graphs for training, validation, as testing, as well as KGs in the known-users (i.e., varying-KG) case and negative interaction examples in the implicit-feedback case. Instead, we assume a simple randomised procedure for generating this information from the benchmark's graphs, and for mitigating the effect of randomness. By this procedure, a benchmark's set $\mathcal{A}$ of interactions is first split into training set $\mathcal{A}_{\mathrm{train}}$, validation set $\mathcal{A}_{\mathrm{val}}$, and test set $\mathcal{A}_{\mathrm{test}}$ as follows. We first randomly select 5% of items in $\mathcal{A}$ as 'unseen', then take all user-item pairs with these items as $\mathcal{A}_{\mathrm{test}}$, and randomly split the rest to $\mathcal{A}_{\mathrm{train}}$ and $\mathcal{A}_{\mathrm{val}}$ with ratio 8:2. Train and validation sets $\mathcal{A}_{\mathrm{train}}$ and $\mathcal{A}_{\mathrm{val}}$ are then given to an evaluated system for training and validation together with KG $\mathcal{K}'$. In the known-users-and-KG case (i.e., when Y is 'fix.'), $\mathcal{K}'$ is the known KG $\mathcal{K}$ of the benchmark (i.e., $\mathcal{K}_{\mathrm{BC}}$, $\mathcal{K}_{\mathrm{ML}}$, $\mathcal{K}_{\mathrm{LF}}$, or $\mathcal{K}_{\mathrm{AB}}$), while in the only known-users case (i.e., when Y is 'var.'), $\mathcal{K}'$ is the sub-KG of $\mathcal{K}$ consisting of all triples of $\mathcal{K}$ not mentioning any 'unseen' item. In order to construct examples for testing, A_{test} is further randomly split, with a ratio 8:2, into two sets $\mathcal{B}_{\mathrm{test}}$ and $\mathcal{L}$. In the implicit-feedback case, an additional 'negative' set $\mathcal{L}^-$ of pairs is sampled using the following strategy: for each user u mentioned in $\mathcal{L}$, we first count the number k of user-item pairs with u in $\mathcal{L}$, then randomly sample k different items $i_1, \ldots, i_k$ from the items of $\mathcal{A}$ such that each (u, i_j) does not belong to $\mathcal{A}$, and finally add all these (u, i_j) to $\mathcal{L}^-$. As a result, the test set in the implicit-feedback case consists of positive examples $(\mathcal{B}_{\mathrm{test}}, \mathcal{K}, \lambda)$ for each $\lambda \in \mathcal{L}$ and negative examples $(\mathcal{B}_{\mathrm{test}}, \mathcal{K}, \lambda)$ for each $\lambda \in \mathcal{L}^-$, where $\mathcal{K}$ is the known KG of the benchmark. The explicit-feedback case is the same except negative examples are not needed. To mitigate the effects of randomness, the

Table 4.2: Benchmark satistics

	BC-X_Y	LF-X_Y	ML-X_Y	AB-X_Y
#users	19,676	1,872	6,040	70,679
#items	20,003	3,846	3,952	24,915
#interactions	172,576	42,346	1,000,209	847,733
#KG entities	25,787	9,366	102,569	88,572
#KG relations	18	60	32	39
#KG triples	60,787	15,518	499,474	2,557,746

benchmarks assume multiple runs of training and testing with independent splitting and sampling, and then reporting the average score and variance of each metric.

4.5.3 Training

Given training and validation sets $\mathcal{A}_{\text{train}}$ and $\mathcal{A}_{\text{val}}$ of interactions in a benchmark together with a KG $\mathcal{K}'$, similar to INDIGO (introduced in Chapter 3), our system InKER is trained on $\mathcal{A}_{\text{train}}$ and $\mathcal{K}'$ as a denoising autoencoder. In particular, $\mathcal{A}_{\text{train}}$ is used, by means of splitting and possibly negative sampling, to construct training examples in the same way as it is done for testing on the base of $\mathcal{A}_{\text{test}}$ (see Section 4.5.2).

We see the task of implicit (explicit) feedback prediction as an ML *classification* (regression) problem. So, we trained InKER and InKER$^-$ using the constructed examples with CE loss for implicit-feedback prediction and Mean Squared Error (MSE) loss for explicit-feedback prediction. In both cases, we trained for 1,500 epochs using the Adam optimiser for each combination of the following hyperparameters: the dimension of vectors in the GNN hidden layers $d \in \{8, 16, 32, 64, 128\}$, the number of GNN layers $L \in \{2, 3, 4\}$, the learning rate $\eta \in \{0.01, 0.005, 0.001\}$ and the L2 penalty $\ell_2 \in \{5 \times 10^{-4}, 5 \times 10^{-5}, 5 \times 10^{-6}, 5 \times 10^{-7}, 5 \times 10^{-8}\}$. Then, based on validation performance, we chose $d = 64$ for BC-$*$, ML-$*$, and AB-$*$, as well as $d = 16$ for LF-$*$; and $L = 2$, $\eta = 0.001$, $\ell_2 = 5 \times 10^{-8}$ for all the benchmarks. We used PyTorch Geometric [43] for the implementation of GNNs underlying InKER variants (i.e., the default GCNs, and optional GATs and GINs). We set the number of attention heads for GAT to 8, and the hyperparameter ϵ for GIN to 0.

To ensure a fair comparison with the baselines, we trained them as described in the corresponding papers and tuned their hyper-parameters in each of the benchmarks using the validation set. For RippleNet, we trained the models on each combination of the following hyperparameters: the hop number $H \in \{2, 3\}$, the dimension of em-

beddings for items and constants in the KGs $d \in \{4, 8, 16, 32\}$, two weights for the regularisation terms $\lambda_1 \in \{10^{-5}, 10^{-6}, 10^{-7}\}$, $\lambda_2 \in \{10^{-2}, 5\times10^{-2}, 10^{-3}\}$, and learning rate $\eta \in \{2 \times 10^{-2}, 10^{-2}, 10^{-3}\}$. For KGCN, the hyperparameters are: the neighbour sampling size $K \in \{4, 8\}$, the dimension of embeddings $d \in \{4, 8, 16, 32\}$, the depth of receptive field $H \in \{1, 2, 3\}$, the weight of L2 regularisation term $\lambda_2 \in \{10^{-4}, 2 \times 10^{-5}, 10^{-7}\}$, and learning rate $\eta \in \{2 \times 10^{-2}, 2 \times 10^{-4}, 5 \times 10^{-4}\}$. For MKR, we trained the models on each combination of the following hyperparameters: the number of low-level layers $L \in \{1, 2, 3\}$, the dimension of embeddings $d \in \{4, 8, 16, 32\}$, the number of times for training on recommendation task $t \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, and the weight for the loss of KGE task $\lambda_1 \in \{0.1, 0.2, 0.5\}$. For RGRec, the hyperparameters are chosen from: the maximum length of rules $I \in \{2, 3, 4\}$, the maximum number of used rules $L \in \{20, 30, 40\}$, and the dimension of embeddings $d \in \{4, 8, 16, 32\}$. For GC-MC, the hyperparameters are chosen from: accumulation function $f_{\text{acc}} = \{\mathsf{stack}, \mathsf{sum}\}$, dropout rate $p_{\text{dropout}} = \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, and the way of normalisation $w_{\text{norm}} = \{\mathsf{left}, \mathsf{symmetric}\}$. For IGMC, the hyperparameters are chosen from: the number of layers for the RGCN $L \in \{2, 3, 4\}$, the dimension of embeddings $d \in \{4, 8, 16, 32\}$, and learning rate $\eta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$. For GraIL, the hyperparameters are chosen from: the number of layers for the GNN $L \in \{2, 3, 4\}$, the dimension of embeddings $d \in \{8, 16, 32\}$, and the hop number $H \in \{1, 2, 3\}$.

The final hyperparameter settings for each of the benchmarks are summarised in Table 4.3. For all the other configurations, we followed the default setting in the original implementation of the baselines.

4.5.4 Metrics

We compared InKER and InKER$^-$ with the relevant baselines on the *implicit-feedback* benchmarks on click-through rate (CTR) prediction and top-k recommendation [3].

For CTR prediction, each system makes a prediction for each user-item pair in the test set; using the corresponding ground truth we compute precision, recall, F1, and ROC-AUC metrics for each system, which are introduced in Section 2.3. (Note that AUC scores rely on various thresholds rather than just one $\theta = 0.5$.)

In the context of top-k recommendation, consider a user u mentioned in the positive test examples. Let S be the set of pairs (i, c), where i is an item in the positive test examples such that (u, i) is not a positive train example, and c is the confidence in the prediction of (u, i). Let then $\mathcal{C}_u^k$ be the items of the first k pairs in S sorted in the decreasing order of c. Then, the *precision@k* metric is the average of $|\mathcal{C}_u^k \cap \mathcal{P}_u|/k$ across all such u, where $\mathcal{P}_u$ is the set of all i such that (u, i) is a positive example;

and the *recall@k* metric is the average of $|\mathcal{C}_u^k \cap \mathcal{P}_u|/|\mathcal{P}_u|$ across all u. The choice of k has a significant impact on these metrics; thus, following Aggarwal[3], we use a combined AUC score based on *precision@k* and *recall@k* for different values of k as our evaluation metric, where we considered all positive integer values of k from 1 to 100. However, too many users can make the experiments time-consuming in practice for several systems; thus, instead of evaluating on all users in positive examples, following the strategy in the original implementation of KGCN, we randomly sample 10 such users. To mitigate the negative effects of random sampling, we compute AUC score 10 times independently, and report the average and variance of the results; we refer to this metric as AUC_k to distinguish it from ROC AUC for CTR prediction.

Table 4.3: Hyperparameter settings for the baselines

		BC-X_Y	LF-X_Y	ML-X_Y	AB-X_Y
RippleNet	$H =$	2	3	2	2
	$d =$	4	32	32	32
	$\lambda_1 =$	10^{-7}	10^{-6}	10^{-7}	10^{-7}
	$\lambda_2 =$	0.01	0.001	0.001	0.001
	$\eta =$	10^{-3}	10^{-2}	2×10^{-2}	2×10^{-2}
KGCN	$K =$	8	8	4	4
	$d =$	4	32	16	32
	$H =$	2	2	3	2
	$\lambda_2 =$	2×10^{-5}	2×10^{-5}	10^{-7}	10^{-7}
	$\eta =$	2×10^{-4}	5×10^{-4}	2×10^{-2}	2×10^{-2}
MKR	$L =$	1	2	1	2
	$d =$	32	8	4	16
	$t =$	6	4	2	2
	$\lambda_1 =$	0.2	0.5	0.2	0.1
RGRec	$I =$	3	3	3	2
	$L =$	30	20	40	20
	$d =$	8	8	8	8
GC-MC	$f_{\text{acc}} =$	sum	sum	stack	sum
	$p_{\text{dropout}} =$	0.6	0.4	0.3	0.3
	$w_{\text{norm}} =$	symmetric	left	left	symmetric
IGMC	$L =$	3	4	3	4
	$d =$	0.6	0.4	0.3	0.3
	$\eta =$	10^{-3}	10^{-3}	10^{-4}	10^{-3}
GraIL	$L =$	3	3	3	3
	$d =$	32	32	16	32
	$H =$	1	3	3	3

We compared InKER and InKER$^-$ with the relevant baselines on the *explicit-feedback* benchmarks using the Normalised Root Mean Square Error (NRMSE) score

Table 4.4: Implicit-feedback results

		BC-Im$_Y$					LF-Im$_Y$				
	Y	AUC	F1	Prec.	Rec.	AUC$_k$	AUC	F1	Prec.	Rec.	AUC$_k$
RippleNet	fix.	.547	.471	.509	.468	.000	.516	.482	.484	.530	.000
KGCN	fix.	.554	.601	.523	.729	.002	.539	.470	.515	.440	.001
KGNN-LS	fix.	.617	.546	.536	.558	.002	.547	.500	.482	.534	.002
MKR	fix.	.433	.358	.445	.303	.000	.486	.483	.459	.607	.000
RGRec	fix.	.484	.629	.509	**.824**	.000	.537	.595	.469	.820	.002
GC-MC	f.&v.	.437	.492	.471	.517	.000	.472	.484	.459	.516	.001
IGMC	f.&v.	.615	.651	.589	.730	.003	.661	.666	.517	.947	.005
GraIL	fix.	.445	.486	.400	.619	.000	.568	.676	.511	**.999**	.003
GraIL	var.	.464	.509	.470	.556	.000	.571	.662	.497	.990	.003
InKER$^-$	f.&v.	.712	.683	**.739**	.636	.004	.742	.680	.585	.871	.011
InKER	fix.	**.756**	**.706**	.705	.699	**.006**	**.784**	.730	**.761**	.710	**.015**
InKER	var.	**.756**	**.706**	.674	.748	**.006**	.779	**.731**	.726	.741	**.015**

		ML-Im$_Y$					AB-Im$_Y$				
	Y	AUC	F1	Prec.	Rec.	AUC$_k$	AUC	F1	Prec.	Rec.	AUC$_k$
RippleNet	fix.	.498	.663	.501	.995	.000	.475	.249	.486	.168	.000
KGCN	fix.	.628	.421	.762	.294	.006	.544	.402	.543	.310	.001
KGNN-LS	fix.	.596	.247	.603	156	.003	.572	.259	.615	.164	.000
MKR	fix.	.498	.622	.501	.840	.000	.461	.247	.444	.173	.000
RGRec	fix.	.504	.409	.329	.542	.000	.503	.628	.475	.925	.000
GC-MC	f.&v.	.497	.479	.496	.467	.001	.503	.546	.508	.594	.000
IGMC	f.&v.	.968	.859	.760	**.998**	.004	.767	.748	.661	.862	.003
GraIL	fix.	.878	.881	.860	.902	.004	.499	.665	.499	.997	.001
GraIL	var.	.845	.848	.830	.867	.003	.501	.665	.500	**.998**	.001
InKER$^-$	f.&v.	.972	.865	**.936**	.804	**.005**	.807	.749	.717	.785	.003
InKER	fix.	.975	.913	.890	.937	**.005**	.810	**.754**	.681	.847	**.004**
InKER	var.	**.976**	**.920**	.912	.928	**.005**	**.813**	.753	**.719**	.791	**.004**

as a metric, which captures the normalised average squared difference between predicted numeric ratings and the corresponding ground truth. In particular, for a set Λ of test user-item pairs (u, i) with ground-truth ratings $y_{u,i}$ and predicted ratings $\hat{y}_{u,i}$, NRMSE is defined as follows, where $y_{\max}$ and $y_{\min}$ are the maximum and the minimum rating in the considered dataset:

$$NRMSE = \frac{1}{y_{\max} - y_{\min}} * \sqrt{\frac{\sum_{(u,i)\in\Lambda} (y_{u,i} - \hat{y}_{u,i})^2}{|\Lambda|}}.$$

Table 4.5: Explicit feedback results (NRMSE)

	Y	BC-Ex$_Y$	LF-Ex$_Y$	ML-Ex$_Y$
GC-MC	f.&v.	.364	–	.324
IGMC	f.&v.	.369	–	.270
InKER$^-$	f.&v.	.361	.077	.270
InKER	fix.	**.360**	**.042**	**.266**
InKER	var.	**.360**	.043	.267

4.5.5 Results

We trained and evaluated InKER and InKER$^-$, as well as RippleNet, KGCN, KGNN-LS, MKR, RGRec, GC-MC, IGMC, and GraIL on each benchmark where it is applicable (note that RippleNet, KGCN, KGNN-LS, MKR, and RGRec can only be applied to $*$-Im$_{\text{fix.}}$ benchmarks, while GraIL to only $*$-Im$_Y$; in turn, GC-MC and IGMC are applicable to all benchmarks except LF-Ex$_Y$ with an unbounded rating domain). The results for implicit- and explicit-feedback recommendation are given in Tables 4.4 and 4.5, respectively, where 'f.&v.' stays for 'both fix. and var.' and is used for the systems that ignore the KG and hence have the same results for both values of Y. Variance values are omitted as they are all lower than 0.005.

Results show that InKER consistently outperforms all the baselines on both CTR prediction and top-k recommendation for both implicit and explicit feedback prediction on all benchmarks (especially on BC-X_Y and AB-X_Y). In particular, we witnessed the superior performance of InKER compared with IGMC, a state-of-the-art inductive recommender system, when neither of the systems utilise any KG information. A possible reason for this advancement is that our approach fully exploits the inductive ability of GNNs, and is more powerful than IGMC (similar to the idea of GraIL) even on the user-item bipartite graph only. Furthermore, there is an improvement on the performance of InKER when incorporating KGs. Compared with GraIL, which is also an inductive system that can take KGs into account, InKER demonstrated better performance. This was anticipated as our KG completion system INDIGO outperforms GraIL (see Chapter 3), and InKER shares a similar idea with INDIGO. Finally, in inductive settings with known users, InKER shows a similar performance to that in the setting with known users and KG.

Table 4.6 and Table 4.7 show the statistics of training time and testing time for all the systems on the benchmarks. In general, InKER achieved a comparable training time to the baselines and is generally faster in testing compared with most of the

Table 4.6: Training time on Implicit Benchmarks (in seconds)

	BC-Im$_{\text{fix.}}$	LF-Im$_{\text{fix.}}$	ML-Im$_{\text{fix.}}$	AB-Im$_{\text{fix.}}$
RippleNet	80.10	46.89	602.50	795.16
KGCN	**9.71**	7.10	283.73	375.58
KGNN-LS	81.75	7.72	236.12	422.17
MKR	16.06	5.10	48.90	**44.52**
RGRec	25.58	13.09	254.11	254.48
GC-MC	22.52	6.15	83.39	118.63
IGMC	27.06	13.41	71.56	353.06
GraIL	131.54	13.67	3440.14	2588.32
InKER	24.38	**5.01**	**37.16**	82.26

Table 4.7: Testing time on Implicit Benchmarks (in seconds)

	BC-Im$_{\text{fix.}}$	LF-Im$_{\text{fix.}}$	ML-Im$_{\text{fix.}}$	AB-Im$_{\text{fix.}}$
RippleNet	0.89	0.26	3.85	6.75
KGCN	**0.33**	0.19	4.74	5.71
KGNN-LS	1.97	0.21	**3.15**	7.20
MKR	0.48	**0.02**	11.30	**3.10**
RGRec	1.86	1.13	144.09	164.14
GC-MC	4.24	0.35	92.71	170.98
IGMC	54.00	1.82	557.74	608.35
GraIL	73.64	6.85	500.00	500.00
InKER	2.88	1.01	171.66	98.39

baselines.

4.5.6 Ablation Study

Following Section 3.5.5, we also conducted ablation studies to investigate the importance of different components. It is worth noting that replacing the encoding approach with a conventional one would result in a system similar to GC-MC, and we have already shown that InKER outperforms GC-MC in Table 4.4 and Table 4.5. Hence, in this section, we further investigated the impact of replacing the GCN used in InKER with a GAT and a GIN variant.

The results are shown in Table 4.8. The scores of GAT- and GIN-based InKER are generally inferior to those of GCN-based, which is consistent with the results in Section 3.5.5. In addition, we witnessed a much slower training for the variants. For

Table 4.8: Ablation results for implicit feedback

		BC-Im$_{\text{var.}}$				LF-Im$_{\text{var.}}$			
	GNN	AUC	F1	Prec.	Rec.	AUC	F1	Prec.	Rec.
InKER	GCN	**.756**	**.706**	**.674**	.748	**.779**	**.731**	.726	**.741**
InKER	GAT	.692	.647	.610	**.766**	.740	.669	**.759**	.603
InKER	GIN	.701	.674	.652	.698	.751	.673	.641	.709

example, training time for GAT- and GIN-based InKER on BC-Im$_{\text{var.}}$ was 37 hours and 18.1 hours, respectively, whereas for InKER with a GCN was only 2.88 hours.

4.5.7 Performance on Sparse Rating Matrices

We have assessed the benefits of using a KG to alleviate the sparsity problem discussed in the introduction. Following the literature [177], we compared InKER$^(-)$ with IGMC, GraIL, KGNN-LS and KGCN on AB-Im$_{\text{var.}}$ by varying the sparsity of the training and validation data. Among these baselines, IGMC is the most powerful baseline nor corporating KGs, and GraIL, KGNN-LS and KGCN all support KGs. In particular, we trained the models on randomly selected 20%, 10%, 5%, 1%, and 0.1% of the interactions in the original training and validation sets, and then tested each of the resulting trained models on the original testing sets of the benchmark.

Our results are summarised in Figure 4.3. As expected, AUC scores of all the systems decrease as the rating matrix in the training set becomes sparser, which indicates the significance of sufficient training data on system performance. We can also observe that InKER outperforms the other baseline in all cases. Furthermore, the performance gap between InKER and InKER$^-$ grows as the sparsity of the training data increases. In addition, with the training data becoming sparser, especially when trained with less than 5% of original training data, the performance of IGMC declines much more sharply than all the other systems, including the systems supporting KGs, and our InKER$^-$ not incorporating KGs. We see these results as evidence that the use of an external KG can help alleviate the sparsity problem in recommender systems, and InKER is still more robust than IGMC even without incorporting KGs.

4.6 Summary

In this chapter, we have proposed a novel approach to inductive recommendation enhanced by KGs. In particular, our approach first encodes the input (interaction

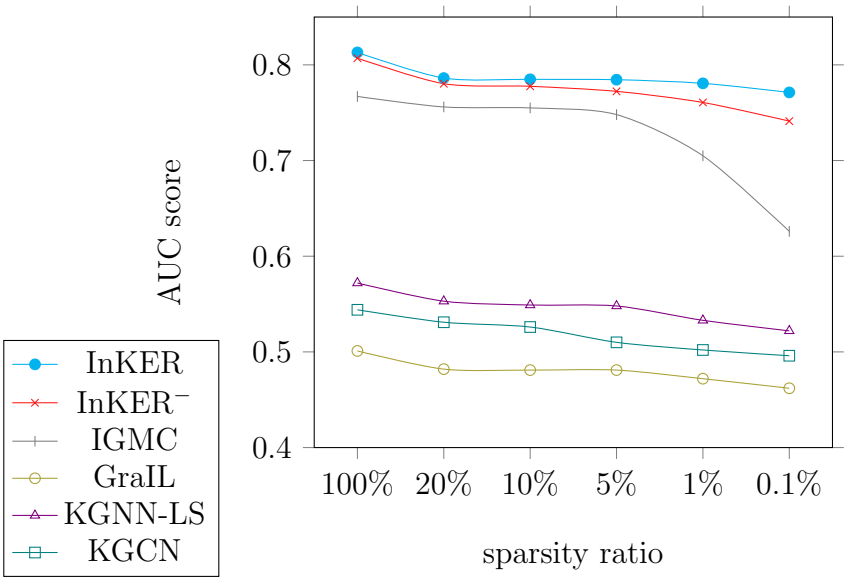


Figure 4.3: Results on rating matrix from AB-Im$_u$ with different sparisity levels

graph, KG, and candidate pairs) as a single graph where nodes are annotated feature vectors. Then, we utilise a GNN model, in particular a GCN, to propagate the feature vectors of nodes in the graph. Finally, a decoder is used to anticipate the scores of candidate user-item pairs from the output of GNN. State-of-the-art systems are limited in that they do not support KGs, support no or limited inductive recommendation, or cannot deal with explicit-feedback ratings; in contrast, our approach imposes no restrictions on the use of KGs or the types of recommendation scenarios, which makes it well-suited for a wide range of applications. Our experiments show that our system outperforms existing baselines on inductive recommendation benchmarks where items are described by an external KG. All in all, we believe this work set out a new perspective to build up a connection among GNN-based recommender systems, KG-enhanced recommender systems, and inductive recommender systems, and can serve as a basis for future work in this direction.

Despite the contributions of our work, there are still some open challenges yet to be addressed, which we leave for future work. First of all, similar to INDIGO, our approach still suffers from the size growth for the pair-wise encoding, which is more crucial for the scenario of recommendation where the amount of data is often very large. Furthermore, in the current version of our work, we omitted the special treatment with types, which can also be very useful information. In addition, our approach fully relies on the structure information of the input graphs. In particular, when testing, there should be at least one fact (either an interaction or a KG triple)

provided for an unseen object, so the unseen objects can not be isolated in the testing graph. This can be potentially resolved by incorporating text or attribute information about the unseen objects. Moreover, our work considers only the external information for items described by KGs, but does not take the user side information into account. Therefore, it would be interesting to utilise both social networks and KGs as side information, and to see if our approach can be further extended to this case. Finally, as discussed in Section 4.1, KGs can help improve the explainability of the recommendation results. Hence, another interesting direction pertains to diving deeper into the explainability problem for the recommender systems, such as extracting the specific part of input (interactions or KG triples) that results in the prediction. Overall, we aspire for our contributions to not only shed light on existing open challenges but also stimulate a broader exploration of novel solutions in this field.

Chapter 5

LogInfer: Inferential Benchmarks for Knowledge Graph Completion Revisited

5.1 Introduction

The availability of suitable benchmarks is key to the development of ML technologies. In this chapter, we further explore the KG completion benchmarks, and, in particular, we consider the transductive settings here, since it is more straightforward, and there are more existing benchmarks to analyse and compare. (In the future, we will also extend it to the inductive settings.) In recent years, a number of benchmarks such as FB15K-237 [128] and WN18RR [36] have become de-facto standards for the evaluation of KG completion models. Positive examples in these benchmarks are obtained by random splitting of the triples in a given real-life KG into training, validation, and test sets; in turn, negative examples are typically generated according to a specific negative sampling method, such as random corruption. The standard benchmarking approach based on random splitting is well-suited for assessing the models' capability to learn a (randomly generated) probability distribution on triples; however, it also comes with significant shortcomings, which we discuss next.

The first important limitation of standard benchmarks is that they provide little information on the models' ability to capture inference patterns introduced in Section 2.4. Examples of such patterns include symmetry, composition, and intersection. There is broad consensus that the ability of KG completion models to learn inference patterns is key to improving the reliability and explainability of their predictions [16].

The ability of KG completion systems to capture inference patterns has been recently analysed from a theoretical perspective [1, 124]. Here, a model captures a pat-

tern if it admits a set of parameters exactly and exclusively satisfying the pattern [1]. For instance, it has been shown that RotatE [124] is able to capture symmetry—that is, given a rule defining a relation as symmetric, there exists a configuration of the models' parameters associated to the relevant relation such that, for any dataset, the models' predictions will coincide with the facts derived by the symmetry rule; in contrast, TransE [18] is not able to capture symmetry in this sense. It is worth noting that this definition of capturing inference patterns is different from the definition for a completion function capturing a rule introduced in Section 3.3.4 and in Proposition 1. The definitions in Abboud et al. [1] and Sun at al. [124] focus more about the theoretical view, but the findings based on such definitions provide little indication of the models' capabilities to learn relevant patterns in practice. In contrast, the definition in Section 3.3.4 considers the models' ability to capture a pattern from a practical perspective, but the limitation is that the results can be affected by some other factors (e.g., if the training is sufficient). Overall, it is important to consider from both the theoretical and the practical perspective. To this end, there is a need for benchmarks that take into account the causal dependencies inherent to inference patterns, thus satisfying the following requirement.

1. During training, models should witness both the premise and conclusion triples for selected rules instantiating a pattern of interest; at the same time, triples used as positive validation and test examples should have supporting evidence in the training set—that is, be the conclusions of rule witnesses with premises in the training set.

Moreover, the second important shortcoming of standard benchmarks lies in their corruption-based strategy for generating negative examples via sampling. As shown by Safavi and Koutra [112], correct classification of corruption-based negative examples is nearly trivial for state-of-the-art KG completion systems. To further illustrate the limitations of corruption-based negative sampling, consider the situation where the function to be learnt involves the dependency formalised by a rule instantiating the intersection pattern:

$$(x, \mathsf{IsParent}, y) \wedge (x, \mathsf{GivesBirth}, y) \rightarrow (x, \mathsf{IsMother}, y); \tag{5.1}$$

however, a particular model learns instead the simpler rule

$$(x, \mathsf{IsParent}, y) \rightarrow (x, \mathsf{IsMother}, y). \tag{5.2}$$

Rule (5.2) logically entails (5.1), but not vice-versa, and hence the model can make (a potentially large number of) wrong predictions. It is worth emphasising that the unintended rule should not be logically entailed by the rules intended for learning. For example, assume that the rule (5.1) and another rule instantiating the hierarchy pattern:

$$(x, \mathsf{GivesBirth}, y) \rightarrow (x, \mathsf{IsParent}, y) \tag{5.3}$$

are expected to be learnt, and another particular model learns the simpler rule

$$(x, \mathsf{GivesBirth}, y) \rightarrow (x, \mathsf{IsMother}, y). \tag{5.4}$$

compared with rule (5.1). However, this model should not be penalised because rule (5.1) and rule (5.3) implies rule (5.4). Corruption-based negative sampling, which are commonly used, would most likely not generate negative examples that penalise the model for learning the unintended rule. To address this issue, we need benchmarks where the negative examples (either included into the benchmark explicitly or produced by a sampling strategy) satisfy the following requirement.

2. Negative training examples should witness the rules that the model should not learn, in particular those that logically entail one of the rules selected for learning, but are not entailed by the rule set selected for training; at the same time, the negative validation and test examples should also include such witnesses, so that a model is penalised for learning unintended rules.

Overall, we call KG completion benchmarks satisfying Requirements 1 and 2 *inferential benchmarks*.

In this section, we propose a novel approach that enables researchers to create inferential benchmarks and assess the performance of their own KG completion systems. The pipeline of our approach for constructing inferential benchmarks starts with a KG and a set of inference patterns of interest, and consists of three main steps. The first step generates rules for the selected inference patterns with a large number of witnessing premises in the KG. The second step applies the rules and distributes the inferred triples amongst sets of positive examples for training, validation, and testing, respectively, so that Requirement 1 is satisfied (the triples in the original KG are also taken as positive training examples). Finally, the third step generates negative training, validation, and testing examples satisfying Requirement 2 by means of some novel negative sampling strategies.

Using our pipeline, we generated a collection of inferential benchmarks based on the common inference patterns and the KGs underpinning FB15K-237 [128],

WN18RR [36], and LUBM [49]. We then conducted a comprehensive evaluation of KG completion systems on these benchmarks, including embedding-based TransE [18], RotatE [124], ComplEx [129], DistMult [164], and BoxE [1]; GNN-based R-GCN [115]; and rule mining AnyBURL [88] and RuleN [89].

Our findings can be summarised as follows.

- All systems performed significantly worse on our benchmarks than on the standard ones, which suggests that benchmarks generated using our approach are challenging for state-of-the-art KG completion systems.
- BoxE and RotatE are the best performing embedding-based models. Rule-based systems outperformed others on simple inference patterns, but not on complex patterns.
- Some models achieved favourable performance despite their theoretical inability to capture certain patterns.
- Performance on classification metrics relies heavily on the choice (or sampling strategy) of negative examples, with our generation methods leading to a considerable performance drop.

These findings highlight the benefits of our benchmarking approach and provide interesting insights for the further development of KG completion methods.

The rest of this section is organised as follows. First of all, in Section 5.2, we will review the existing knowledge graph completion benchmarks. Then, we will introduce our pipeline for constructing inferential benchmarks satisfying the two requirements in Section 5.3. Furthermore, we will present a series of benchmarks generated following this pipeline in Section 5.4 and a comprehensive empirical evaluation for state-of-the-art KG completion systems on these benchmarks in Section 5.5. The conclusion and future work will be discussed in Section 5.6.

5.2 Related Work

In this section, we will briefly introduce the existing KG completion benchmarks in the transductive settings, which can be generally classified into benchmarks that are not based on inference patterns, and that are based on inference patterns.

For benchmarks that are not based on inference patterns, they often rely on random splitting of triples in real-life KGs to construct positive examples, and a conventional random corruption strategy to generate negative examples. Concrete examples include FB15K [18] and WN18 [18], and their variants FB15K-237 [128] and WN18RR [36] where all the reverse relations are removed.

KG completion benchmarks based on inference patterns are designed to assess systems' capability to capture and predict specific inference patterns within a KG. Benchmarks Kinship [64] and Country [19] are based on simple inference patterns and involve datasets of small size with a very limited number of relations. For example, the Country benchmark is based on a KG describing all the ground truth location information about 244 countries: the subcontinent (e.g. Western Europe) and the continent (e.g. Europe) that each country (e.g. France) is located in. It first splits the countries into 204 countries for training, 20 countries for validation, and 20 countries for testing. The positive validation set consists of all the triples of the form $(c_{\text{valid}}, \mathsf{locatedIn}, r_{\text{valid}})$, where c_{valid} is a country for validation, and r_{valid} is a continent. The positive test set consists of all the facts of the form $(c_{\text{test}}, \mathsf{locatedIn}, r_{\text{test}})$ where c_{test} is a country for testing, and r_{test} is a continent. The positive training set consists of all the other triples in the KG. As we can see, in the training set, we can observe the location information with triples $(c_{\text{train}}, \mathsf{locatedIn}, s_{\text{train}})$ and $(c_{\text{train}}, \mathsf{locatedIn}, r_{\text{train}})$ where c_{train} is a country for training, s_{train} is a subcontinent and r_{train} is a continent. Moreover, we can also see the location information of all the subcontinents and continents with triples of the form $(s_{\text{train}}, \mathsf{locatedIn}, r_{\text{train}})$. Hence, during training, it can witness both premise and conclusions of a rule

$$(x, \mathsf{locatedIn}, y) \wedge (y, \mathsf{locatedIn}, z) \rightarrow (x, \mathsf{locatedIn}, z). \tag{5.5}$$

Moreover, for each country for testing, we can observe in the training set the triples about which subcontinent it is located in, and which continent this subcontinent is located in. Therefore, each triple in the positive testing set has supporting evidence in the training set, and thus the benchmark satisfies our Requirement 1 mentioned in Section 5.1. Similarly, the Kinship benchmark also satisfies Requirement 1. However, these benchmarks fail to satisfy Requirement 2 as they do not include negative examples and (silently) rely on the corruption-based negative sampling strategy.

Cao et al. [23] recently proposed a more sophisticated benchmark InferWiki based on Wikidata. Their approach relies on the rule mining system AnyBURL [88] for generating relevant rules of a very specific syntactic shape. Triples witnessing the premises of these rules are included in the training set as positive examples, while all

the conclusion triples are included as positive examples in the test set; as a result, InferWiki does not satisfy Requirement 1 since models are not able to witness during training both the premise and the conclusion triples for the selected rules. Candidate negative examples are generated using the conventional random corruption strategy and are then subsequently filtered by human annotators on the base of their plausibility in real life; thus, InferWiki does not satisfy Requirement 2 either. Overall, we can draw a conclusion that the existing benchmarks based on inferential patterns do not satisfy all our requirements for inferential benchmarks.

There is also rich literature discussing the issues with existing benchmarks and evaluation protocols. For instance, some papers [7, 23, 146] argue that the negative examples in the benchmarks are unchallenging in nature, and they propose to sample negative examples based on the statistics or types of the constants, or the exclusivity of the relations. In addition, Akrami et al. [4] analysed systematically the leakage issue of the test set, and conducted experiments to show its impact on the system performance. Finally, the potential unfairness of ranking-based metrics has also garnered considerable attention [15, 125].

In the subsequent sections, we will propose a benchmarking approach for generating benchmarks satisfying the two requirements discussed in Section 5.1, and will also devise several strategies for generating more challenging negative examples.

5.3 Inferential Benchmark Construction

In this section, we introduce our pipeline for constructing KG completion benchmarks satisfying the two key requirements postulated in the introduction. An inferential benchmark is built from a KG $\mathcal{K}$ and one or several inference patterns $\mathfrak{P}$. We assume that the rest of this section is based on the preliminaries defined in Chapter 2, and in this work we focus on the *transductive* KG completion problem. It is worth noting that, following in the definition in Section 2.2, the positive sets $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$ are assumed to be positive training, validation, and test examples in the transductive settings, and we have $\mathcal{P}_{\text{train}} = P_{\text{train}}$, $\mathcal{P}_{\text{valid}} = P_{\text{valid}}$, and $\mathcal{P}_{\text{test}} = P_{\text{test}}$. This is analogous to the negative cases. Our approach for constructing an inferential benchmark for $\mathcal{K}$ and $\mathfrak{P}$ then consists of three steps:

1. rule generation, which produces a set $\mathcal{R}$ of rules based on $\mathcal{K}$ and $\mathfrak{P}$ so that the size of their support (i.e., the number of witnessing premises) in $\mathcal{K}$ is maximised;

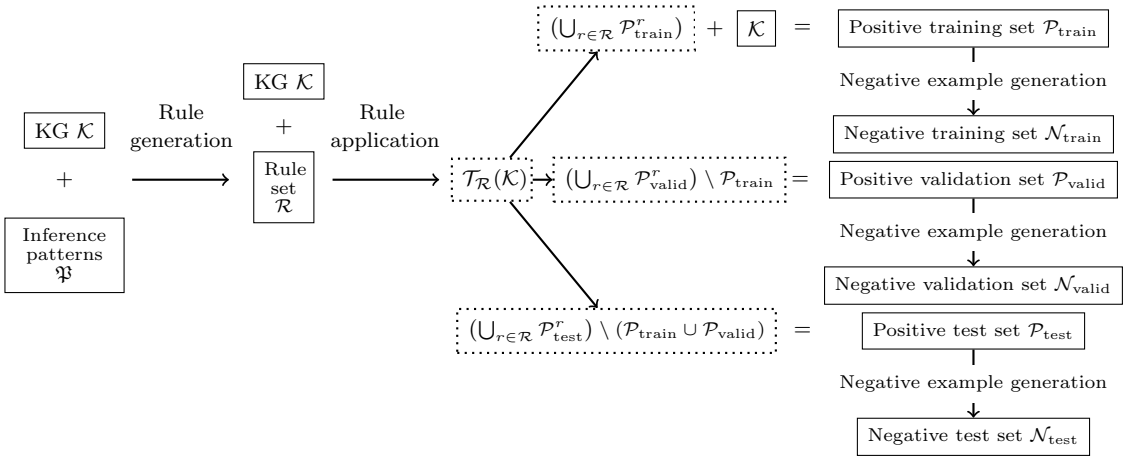


Figure 5.1: Our pipeline for constructing an example inferential benchmark

2. rule application and distributing the results, which distributes the one-step application result $\mathcal{T}_{\mathcal{R}}(\mathcal{K})$ and $\mathcal{K}$ itself into the training, validation, and test positive sets $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$, so that Requirement 1 is satisfied;

3. negative example generation, which constructs appropriate and challenging negative examples according to one of three negative sampling methods which we will introduce in Section 5.3.3, so that Requirement 2 is satisfied.

A summary of our pipeline is depicted in Figure 5.1, and a simple example benchmark is given in Figure 5.2. As we can see, the triples (Alex, IsColleague, Bob), (Bob, IsColleague, John) and (Alex, IsColleague, John) can be seen as a support of the rule $(x, \mathsf{IsColleague}, y) \wedge (y, \mathsf{IsColleague}, z) \rightarrow (x, \mathsf{IsColleague}, z)$. Similarly, we have (Harry, IsColleague, James), (James, IsColleague, Tony) and (Harry, IsColleague, Tony) as another support in the positive training set. With the (Ada, IsColleague, Lucy) as a positive testing triple, we can find its supporting evidence (Ada, IsColleague, Eve) and (Eve, IsColleague, Lucy) in the positive training set.

5.3.1 Rule Generation

We initiate rule generation by constructing a set $\mathcal{R}_{\text{cand}}$ of candidate rules from inference patterns $\mathfrak{P}$ (and $\mathsf{Preds}(\mathcal{K})$). First, for each pattern in $\mathfrak{P}$ with the head predicate (type or relation) template mentioned in the body, $\mathcal{R}_{\text{cand}}$ contains each rule obtained from the pattern by substituting all predicate templates by predicates in $\mathsf{Preds}(\mathcal{K})$. Second, for each pattern in $\mathfrak{P}$ with the head template not mentioned in the body, $\mathcal{R}_{\text{cand}}$ contains one rule for every substitution of the templates in the body as above; in turn, the head template is substituted by a random type or relation from $\mathsf{Types}(\mathcal{K})$

Positive training set:

(Alex, IsColleague, Bob), (Bob, IsColleague, John),
(Alex, IsColleague, John), (Harry, IsColleague, James),
(James, IsColleague, Tony), (Harry, IsColleague, Tony),
(Ada, IsColleague, Eve), (Eve, IsColleague, Lucy)

Negative training set:

(Alex, IsColleague, James), (Alex, IsColleague, Tony),
(Alex, IsColleague, Eve), (Harry, IsColleague, John),
(James, IsColleague, John), (Ada, IsColleague, John),

Positive test set:	Negative test set:
(Ada, IsColleague, Lucy)	(Ada, IsColleague, James)

Figure 5.2: An example inferential benchmark based on the rule $(x, \mathsf{IsColleague}, y) \wedge (y, \mathsf{IsColleague}, z) \rightarrow (x, \mathsf{IsColleague}, z)$ and the position-aware corruption method for negative example generation (validation set is omitted for simplicity)

and $\mathsf{Rels}(\mathcal{K})$, respectively. This is justified by the fact that rules differing only in the head predicate are essentially equivalent for learning purposes.

Although set $\mathcal{R}_{\text{cand}}$ may be very large, the majority of its rules are not useful for generating examples as they do not apply to $\mathcal{K}$ sufficiently many times. So, we complete the rule generation step by selecting a subset $\mathcal{R}$ of rules $\mathcal{R}_{\text{cand}}$ with large support in $\mathcal{K}$. One approach is to select into $\mathcal{R}$ a fixed predefined number of relations with the largest support; note, however, that we do this separately for each pattern to ensure that each of them is represented and can be learned—that is, for each pattern, we include k_1 rules with the largest support in $\mathcal{K}$, where k_1 is a predefined number that can be customised based on the expected size and rule diversity of the benchmark. The support can be computed using a SPARQL engine (we used RDFox [98], see Section 5.5). Selecting rules with the largest support is, however, not essential, and an alternative approach is to manually select rules with large enough support. We will use both approaches in our benchmarks.

Instead of using rule mining models to generate rules from the KG, which only generate very specific types of rules, we choose to generate rules more randomly. As a result, we may generate rules that do not make sense from a modelling perspective, such as $(x, \mathsf{Speaks}, y) \rightarrow (x, \mathsf{LocatedIn}, y)$. This is justified by that fact that our aim is to design benchmarks that test the ability to learn rules according to inference patterns, rather than according to modelling considerations.

Table 5.1: Inference patterns considered in this work

	Pattern
Symmetry	$(x, \_\mathsf{R}, y) \rightarrow (y, \_\mathsf{R}, x)$
Inversion	$(x, \_\mathsf{R}, y) \rightarrow (y, \_\mathsf{S}, x)$
Hierarchy	$(x, \_\mathsf{R}, y) \rightarrow (x, \_\mathsf{S}, y)$
Composition	$(x, \_\mathsf{R}, y) \wedge (y, \_\mathsf{S}, z) \rightarrow (x, \_\mathsf{T}, z)$
Intersection	$(x, \_\mathsf{R}, y) \wedge (x, \_\mathsf{S}, y) \rightarrow (x, \_\mathsf{T}, y)$
Triangle	$(x, \_\mathsf{R}, y) \wedge (x, \_\mathsf{S}, z) \wedge (y, \_\mathsf{T}, z) \wedge (x \neq y)$ $\wedge (x \neq z) \wedge (y \neq z) \rightarrow (x, \_\mathsf{P}, y)$
Diamond	$(x, \_\mathsf{R}, y) \wedge (x, \_\mathsf{S}, z) \wedge (y, \_\mathsf{T}, w) \wedge (z, \_\mathsf{P}, w)$ $\wedge (x \neq y) \wedge (x \neq z) \wedge (x \neq w) \wedge (y \neq z)$ $\wedge (y \neq w) \wedge (z \neq w) \rightarrow (x, \_\mathsf{Q}, y)$

5.3.2 Distributing Rule Application Results

The second step constructs the positive examples in $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$ for training, validation, and testing, respectively. To satisfy our Requirement 1, we include the original KG $\mathcal{K}$ in $\mathcal{P}_{\text{train}}$ and then distribute the triples in $\mathcal{T}_{\mathcal{R}}(\mathcal{K})$, obtained by applying the selected rules $\mathcal{R}$ to $\mathcal{K}$, between the three sets $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$ as described next.

To avoid data leakage (i.e., the situation where test examples are observed during training), we only consider the newly derived triples for distribution. Moreover, we distribute these triples independently rule by rule to ensure that each rule can be learned, at the same time ensuring that the same triple does not end up in more than one of the three sets (this must be done with care since a triple may be derived by several different rules); however, to ensure a reasonable size of the dataset, we sample a fixed number of triples for each of the rules before distribution.

We first compute $\mathcal{T}_r(\mathcal{K}) \setminus \mathcal{K}$ for each rule $r \in \mathcal{R}$, then randomly sample up to k_2 triples from $\mathcal{T}_r(\mathcal{K}) \setminus \mathcal{K}$ (where k_2 is again a predefined number that can be customised based on the expected size), and split the sampled triples into three sets, $\mathcal{P}^r_{\text{train}}$, $\mathcal{P}^r_{\text{valid}}$, $\mathcal{P}^r_{\text{test}}$, according to a predefined ratio (e.g., 8:1:1 in our benchmarks, see Section 5.5). Then, we take

$$\mathcal{P}_{\text{train}} = \left(\bigcup\nolimits_{r \in \mathcal{R}} \mathcal{P}^r_{\text{train}}\right) \cup \mathcal{K}$$

as the positive training set,

$$\mathcal{P}_{\text{valid}} = \left(\bigcup\nolimits_{r \in \mathcal{R}} \mathcal{P}^r_{\text{valid}}\right) \setminus \mathcal{P}_{\text{train}}$$

as the positive example triples for validation, and

$$\mathcal{P}_{\text{test}} = \left(\bigcup_{r \in \mathcal{R}} \mathcal{P}^r_{\text{test}}\right) \setminus (\mathcal{P}_{\text{train}} \cup \mathcal{P}_{\text{valid}})$$

as the positive example triples for testing. Finally, we again let $\mathcal{P}_{\text{all}} = \mathcal{P}_{\text{train}} \cup \mathcal{P}_{\text{valid}} \cup \mathcal{P}_{\text{test}}$.

Consider an example of our benchmark in Figure 5.2, during training, a model could witness (Alex, IsColleague, Bob), (Bob, IsColleague, John) as premises, and witness (Alex, IsColleague, John) as conclusion, and a similar witness is enabled for Harry, James, and Tony. In addition, for the positive test example (Ada, IsColleague, Lucy), its premises (Ada, IsColleague, Eve) and (Eve, IsColleague, Lucy) are included in the training set. Therefore, our benchmark satisfies the Requirement 1 mentioned in the introduction.

5.3.3 Negative Example Generation

The third step of our approach tackles the generation of negative examples $\mathcal{N}_{\text{train}}$, $\mathcal{N}_{\text{valid}}$, and $\mathcal{N}_{\text{test}}$ for training, validation, and testing, respectively, so that Requirement 2 for inferential benchmarks is satisfied. In particular, negative examples are generated so as to witness rules that the model should not learn, especially those that logically entail one rule from $\mathcal{R}$, but are not logically entailed by the program $\mathcal{R}$. In contrast to standard benchmarks, which provide a (corruption-based) negative sampling strategy, benchmarks produced by our approach do include concrete negative examples; this allows us to specify more precisely the undesired dependencies that should be prevented from learning, and hence make negative examples more challenging to classify.

We introduce three methods for generating negative examples: relevance-based sampling, position-aware corruption, and query-guided sampling. All three methods ensure balance between positive and negative examples—that is, $|\mathcal{N}_{\text{train}}| = |\mathcal{P}_{\text{train}}|$, $|\mathcal{N}_{\text{valid}}| = |\mathcal{P}_{\text{valid}}|$, and $|\mathcal{N}_{\text{test}}| = |\mathcal{P}_{\text{test}}|$.

Relevance-based sampling. *Relevance-based sampling* relies on random generation of negative examples involving predicates from the heads of rules $\mathcal{R}$, and entities from the triples of $\mathcal{K}$ contributing to the support of these rules (i.e., matching the rules' bodies); additionally, it is ensured that the generated examples are truly negative—that is, not mentioned in any of the positive sets. Formally, let $\mathsf{Pred}_{\mathcal{R}}$ be the set of predicates in the heads of the rules in $\mathcal{R}$, $\mathsf{Const}_{\text{sup}}$ be all the constants from $\mathcal{K}$ occurring in the support of rules in $\mathcal{R}$, and $\mathcal{N}_{\text{cand}}$ be the set of triples over $\mathsf{Pred}_{\mathcal{R}}$ and

$\mathsf{Const}_{\text{sup}}$. Then, we take $\mathcal{N}_{\text{train}}$, $\mathcal{N}_{\text{valid}}$, and $\mathcal{N}_{\text{test}}$ as disjoint sets of the size specified above randomly sampled from $\mathcal{N}_{\text{cand}} \setminus \mathcal{P}_{\text{all}}$ without repetition. Note that this simple method is likely to ensure Requirement 2, because the predicates and constants of such negative examples are all involved in rule applications generating positive examples, and so it is likely that they represent rules that are similar to $\mathcal{R}$ but should not be learned.

Position-aware corruption. *Position-aware corruption* is a more fine-grained approach than relevance-based sampling. In particular, instead of sampling from all unseen triples in $\mathcal{N}_{\text{cand}}$, sampling is restricted to conclusion triples corrupted with constants seen in a similar context. Formally, let $\mathcal{C}_{\text{train}}$, $\mathcal{C}_{\text{valid}}$, and $\mathcal{C}_{\text{test}}$ be the sets of conclusion triples in $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$, respectively, for rules $\mathcal{R}$ in $\mathcal{K}$ (note that, by construction, $\mathcal{C}_{\text{valid}} = \mathcal{P}_{\text{valid}}$ and $\mathcal{C}_{\text{test}} = \mathcal{P}_{\text{test}}$, but we give them different names for uniformity). Let $\mathcal{C}'_{\text{train}}$ be the set that contains

- each triple $(e', \mathsf{type}, t) \notin \mathcal{P}_{\text{all}}$ such that there exist triples $(e, \mathsf{type}, t) \in \mathcal{C}_{\text{train}}$ and $(e', \mathsf{type}, t') \in \mathcal{P}_{\text{all}}$;
- each triple $(s', R, o) \notin \mathcal{P}_{\text{all}}$ such that there exist triples $(s, R, o) \in \mathcal{C}_{\text{train}}$ and $(s', R, o') \in \mathcal{P}_{\text{all}}$; and
- each triple $(s, R, o') \notin \mathcal{P}_{\text{all}}$ such that there exist triples $(s, R, o) \in \mathcal{C}_{\text{train}}$ and $(s', R, o') \in \mathcal{P}_{\text{all}}$.

Moreover, let $\mathcal{C}'_{\text{valid}}$ and $\mathcal{C}'_{\text{test}}$ be constructed in the same way from $\mathcal{C}_{\text{valid}}$ and $\mathcal{C}_{\text{test}}$, respectively. Finally, we take $\mathcal{N}_{\text{train}}$, $\mathcal{N}_{\text{valid}}$, and $\mathcal{N}_{\text{test}}$ as sets of the sizes as specified above randomly sampled from $\mathcal{C}'_{\text{train}}$, $\mathcal{C}'_{\text{valid}} \setminus \mathcal{N}_{\text{train}}$, and $\mathcal{C}'_{\text{test}} \setminus (\mathcal{N}_{\text{train}} \cup \mathcal{N}_{\text{valid}})$, respectively, without repetition (note that the set difference is taken to avoid data leakage). This simple method makes it even more likely that the predicates and constants represent undesired rules similar to $\mathcal{R}$, and hence makes further progress towards ensuring our Requirement 2 for inferential benchmarks.

Query-guided sampling. *Query-guided sampling* refines position-aware corruption and puts more emphasis on preventing systems from learning the simple rule that entail one of the rules in $\mathcal{R}$, but not logically entailed by $\mathcal{R}$. (e.g., Rule (5.1) follows from Rule (5.2) and so, assuming that (5.1) is in $\mathcal{R}$, we should ensure that (5.2) is not learned, unless it logically follows from the rules in $\mathcal{R}$). Since there is usually an infinite number of rules logically entailing another rule, we take a pragmatic approach and, for the purpose of generating negative examples, concentrate on rules obtained from rules in $\mathcal{R}$ by taking a subset of their body atoms, and check for each of the

rules if it is entailed by the rule set $\mathcal{R}$. To cover rules that do not have any sub-rules entailing them, including those with a single body atom, we also generate negative examples using the position-aware corruption method. Formally, let $\mathcal{R}^-$ be the set of rules that each rule in the set is obtained from a rule in $\mathcal{R}$ by removing one or more body atoms, and is not logically entailed by $\mathcal{R}$. Let $\mathcal{R}_{\text{complex}}$ be the rules in $\mathcal{R}$ that contribute to this process. Then, let $\mathcal{C}^-_{\text{train}}$, $\mathcal{C}^-_{\text{valid}}$, and $\mathcal{C}^-_{\text{test}}$ be a split of the set $\mathcal{T}_{\mathcal{R}^-}(\mathcal{K}) \setminus \mathcal{P}_{\text{all}}$ with the same ratio as we used in Section 5.3.2. Finally, we take $\mathcal{N}_{\text{train}}$, $\mathcal{N}_{\text{valid}}$, and $\mathcal{N}_{\text{test}}$ as sets of the sizes as specified above constructed as follows: up to $|\mathcal{R}_{\text{complex}}|/|\mathcal{R}|$ fraction of needed triples is sampled from $\mathcal{C}^-_{\text{train}}$, $\mathcal{C}^-_{\text{valid}}$, and $\mathcal{C}^-_{\text{test}}$ without repetition, and the rest is sampled from $\mathcal{C}'_{\text{train}}$, $\mathcal{C}'_{\text{valid}} \setminus \mathcal{N}_{\text{train}}$, and $\mathcal{C}'_{\text{test}} \setminus (\mathcal{N}_{\text{train}} \cup \mathcal{N}_{\text{valid}})$, respectively, where $\mathcal{C}'_{\text{train}}$, $\mathcal{C}'_{\text{valid}}$, and $\mathcal{C}'_{\text{test}}$ are defined as in the position-aware case (note that 'up to' in the first sampling is essential because $\mathcal{C}^-_{\text{train}}$, $\mathcal{C}^-_{\text{valid}}$, and $\mathcal{C}^-_{\text{test}}$ may not have sufficiently many triples; however, the sampling here adheres as much as possible to the specified bound).

5.4 Benchmarks

Following our methodology proposed in Section 5.3, we have constructed a suite of 37 benchmarks built upon three KGs: those underpinning the standard benchmarks FB15K-237 [128] and WN18RR [36], denoted as $\mathcal{K}_{\text{fb}}$ and $\mathcal{K}_{\text{wn}}$, and the synthetic KG LUBM(1,0) [49], denoted as $\mathcal{K}_{\text{lubm}}$. We used RDFox [98] SPARQL engine to compute $\mathcal{T}_{\mathcal{R}}(\mathcal{K})$ triples and in other similar cases.

Each benchmark based on $\mathcal{K}_{\text{fb}}$ and $\mathcal{K}_{\text{wn}}$ aims to test systems' ability to learn a single inference pattern, and we concentrate on the patterns in Table 5.1. The intersection pattern on $\mathcal{K}_{\text{wn}}$ does not give a large-enough number of positive examples, so we omitted this case; we also note that query-guided negative sampling is relevant only to the intersection, triangle, and diamond patterns. So we constructed 31 benchmarks, and we use the notation LogInfer-X^Y_Z, where $X \in \{\text{FB}, \text{WN}\}$ specifies the KG, $\mathcal{K}_{\text{fb}}$ or $\mathcal{K}_{\text{wn}}$, $Y \in \{\text{sym}, \text{inver}, \text{hier}, \text{comp}, \text{inter}, \text{trian}, \text{diam}\}$ specifies the pattern in Table 5.1, and $Z \in \{\text{rb}, \text{pa}, \text{qg}\}$ specifies the negative example generation method (with $Y = \text{inter}$ and $Z = \text{qg}$ applicable to only some cases as described above). We used 8:1:1 as the splitting ratio, and took numbers k_1 and k_2 as specified in Table 5.2 to ensure appropriate size and variety.

The benchmarks based on $\mathcal{K}_{\text{lubm}}$ rely on the 107 rules provided by Nenov et al. [98]. These rules are designed to have large support, and hence are well-suited for generating sets of positive examples; furthermore, they instantiate a wide range

Table 5.2: Benchmark statistics, where each number applies for all relevant Z (the sizes of the negative example sets $|\mathcal{N}_{\text{train}}|$, $|\mathcal{N}_{\text{valid}}|$, and $|\mathcal{N}_{\text{test}}|$ are the same as for positive examples)

	Y	k_1	k_2	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{P}_{\text{train}}\rvert$	$\lvert\mathcal{P}_{\text{valid}}\rvert$	$\lvert\mathcal{P}_{\text{test}}\rvert$
LogInfer-FB^Y_Z	sym	50	200	310,116	318,116	1,000	1,000
	inver	200	200	310,116	341,603	3,928	3,946
	hier	200	200	310,116	341,477	3,915	3,934
	inter	200	200	310,116	333,478	2,908	3,126
	comp	200	200	310,116	341,829	3,964	3,966
	trian	200	200	310,116	341,974	3,975	3,975
	diam	200	200	310,116	340,625	3,803	3,822
LogInfer-WN^Y_Z	sym	5	2000	93,003	101,003	1,000	1,000
	inver	5	2000	93,003	101,003	1,000	1,000
	hier	5	2000	93,003	101,003	1,000	1,000
	comp	20	2000	93,003	114,336	1,610	1,564
	trian	20	2000	93,003	100,800	968	977
	diam	20	2000	93,003	104,681	1,398	1,391
LogInfer-LUBM^Y_Z	all	107	500	103,119	140,536	5,001	5,003
	no-type	19	2500	103,119	140,540	4,999	5,002

of inference patterns, including symmetry, hierarchy, inversion, and composition. We consider two variants of these benchmarks: one that uses all the rules, and one that uses only the 19 rules not mentioning any types. The latter is justified by the fact that many KG completion systems do not have special treatment for type triples and consider them as regular triples over the type 'relation'; this leads to significantly poorer performance in comparison to approaches with dedicated care of type triples [93, 158]. Overall, we constructed 6 benchmarks based on LUBM denoted as LogInfer-LUBM^Y_Z, where $Y \in \{\text{all}, \text{no-type}\}$ specifies whether the rules with types are included or not, and $Z \in \{\text{rb}, \text{pa}, \text{qg}\}$ specifies the negative example generation method. We used the ratio 8:1:1, and k_1 and k_2 as in Table 5.2.

The statistics of constructed benchmarks are summarised in Table 5.2. The benchmarks themselves and the accompanying documentation are available online.[1]

5.5 Evaluation

We have evaluated a representative sample of eight state-of-the-art KG completion systems on the benchmarks described in Section 5.4.

[1]https://github.com/shuwen-liu-ox/LogInfer

5.5.1 Systems and Training

The evaluated systems can be divided into the following three categories:

1. *embedding-based* methods, including TransE [18], RotatE [124], ComplEx [129], DistMult [164], and BoxE [1];
2. *GNN-based* methods, including R-GCN [115]; and
3. *rule mining* methods, including AnyBURL [88] and RuleN [89].

In the case of TransE, RotatE, ComplEx, and DistMult, we used the implementations provided by Sun et al. [124]; for the other systems, we used the implementations provided by the respective authors. Additionally, we have implemented a simple baseline SimpBL, which predicts a test triple (a, b, c) as true if and only if $\mathcal{P}_{\text{train}}$ contains a triple involving both a and b, and a triple involving both b and c.

All the evaluated state-of-the-art systems provide confidence values in $[0, 1]$ for each prediction; the threshold needed for computing classification-based metrics is therefore considered a hyperparameter optimised during validation. In contrast, SimpBL outputs Boolean values and hence metrics relying on confidence predictions are not applicable.

5.5.2 Results

We have evaluated all systems on all the benchmarks described in Section 5.4. However, due to the large number of benchmarks and metrics, we report only a representative selection of the obtained results. In particular, we report (ROC) AUC, F1, precision, recall, C-MRR and R-MRR, and concentrate on query-guided method where applicable and position-aware corruption in the remaining cases. The results for LogInfer-FB^Y_Z and LogInfer-WN^Y_Z, where $Z = \text{pa}$ for $Y \in \{\text{sym, inver, hier, comp}\}$, and $Z = \text{qg}$ for $Y \in \{\text{inter, trian, diam}\}$ are given in Table 5.3 and Table 5.4; in turn, the results for LogInfer-LUBM$^{\text{all}}_{\text{qg}}$ and LogInfer-LUBM$^{\text{no-type}}_{\text{qg}}$ are in Table 5.5. In the tables, 'Prec.', 'Rec.', 'R-M.', and 'C-M.' stay for precision, recall, R-MRR, and C-MRR, respectively. Our results can be summarised as follows.

1. All systems clearly and consistently outperformed our simple baseline on all benchmarks, with BoxE and RotatE outperforming all other embedding-based methods.

2. Systems' performance was significantly better across the board for simple inference patterns (i.e., symmetry, inversion, and hierarchy) than for more complex patterns involving conjunctions and inequalities in bodies (i.e., composition, triangle, and diamond).

3. Rule-based systems exhibited better performance than the other systems on simple patterns; however, the gain is not significant on complex patterns. This can be attributed to the fact that rule-based systems, on the one hand, mine rules based on the support of their bodies in the training set but, on the other hand, do not exploit the negative training examples, which penalise systems for learning unintended rules.

4. Some of our results are well-aligned with the theoretical findings on the expressive power of KG completion models; for instance, TransE and DistMult performed poorly on patterns that they cannot capture theoretically (namely symmetry and inversion, respectively). In contrast, other models achieved good performance on patterns that they cannot theoretically capture; for example, RotatE cannot capture hierarchy, but showed strong performance on LogicInfer-$\text{WN}^{\text{hier}}_{\text{pa}}$ and LogicInfer-$\text{FB}^{\text{hier}}_{\text{pa}}$. A possible reason is that RotatE captured a more specific pattern: it can capture $(x, R, y) \rightarrow (x, S, y)$ provided the embeddings of R and S coincide, in which case $(x, S, y) \rightarrow (x, R, y)$ also holds. By making predictions using $(x, R, y) \leftrightarrow (x, S, y)$, RotateE may perform well.

5. Relative performance varied across different metrics; for instance, R-GCN generally outperforms other models on R-MRR, but fares worse on C-MRR; this can be explained by the design of R-GCN, which learns relation-specific parameters and so is adept at distinguishing relations.

5.5.3 Impact of Negative Example Generation

We have studied the impact of the different negative example generation methods on LogInfer-$\text{FB}^{\text{sym}}_{Z}$ for $Z \in \{\text{rc}, \text{rb}, \text{pa}\}$ and LogInfer-$\text{FB}^{\text{inter}}_{Z}$ for $Z \in \{\text{rc}, \text{rb}, \text{pa}, \text{qg}\}$; here, $Z = \text{rc}$ corresponds to the conventional random object corruption method where, for each triple (s, R, o) in $\mathcal{P}_{\text{train}}$, $\mathcal{P}_{\text{valid}}$, and $\mathcal{P}_{\text{test}}$, the corresponding $\mathcal{N}_{\text{train}}$, $\mathcal{N}_{\text{valid}}$, and $\mathcal{N}_{\text{test}}$ contains a triple $(s, R, o') \notin \mathcal{P}_{\text{all}}$ for o' randomly sampled from $\mathsf{Consts}(\mathcal{K})$ in a way that the negative sets remain disjoint.

Table 5.6 summarises our results for the AUC metric. As we can see, systems achieved high scores whenever the conventional random corruption method was used

Table 5.3: Evaluation results on LogInfer-FB$^Y_{\text{pa}}$ and LogInfer-WN$^Y_{\text{pa}}$ (in %) with $Y \in \{\text{sym}, \text{inver}, \text{hier}, \text{comp}\}$.

		LogInfer-FB$^Y_{\text{pa}}$						LogInfer-WN$^Y_{\text{pa}}$					
Y	Model	AUC	F1	Prec.	Rec.	R-M.	C-M.	AUC	F1	Prec.	Rec.	R-M.	C-M.
sym	SimpBL	-	66.7	50.0	**100.0**	-	-	-	30.5	31.8	29.4	-	-
	TransE	47.0	56.2	45.9	72.3	9.1	3.4	98.8	98.8	98.6	99.0	45.1	31.3
	RotatE	86.8	93.1	87.7	99.1	56.8	19.9	99.1	99.1	98.5	99.8	50.0	61.6
	ComplEx	73.0	83.5	74.0	95.8	48.3	16.3	80.3	81.1	77.8	84.8	49.4	6.9
	DistMult	97.5	98.6	97.7	99.5	**89.7**	41.7	82.3	83.1	79.5	87.0	50.4	15.1
	BoxE	92.8	94.9	95.3	94.6	87.3	30.6	**100.0**	**100.0**	**100.0**	**100.0**	53.6	**68.8**
	R-GCN	92.8	96.2	92.8	**100.0**	81.0	17.4	96.1	96.2	92.8	**100.0**	**81.0**	54.7
	AnyBURL	96.0	97.9	96.0	**100.0**	86.3	43.6	99.9	99.9	99.9	**100.0**	53.5	63.7
	RuleN	**98.0**	**99.0**	**98.0**	**100.0**	86.6	**43.7**	88.3	86.7	**100.0**	76.5	53.8	51.1
inver	SimpBL	-	36.2	38.2	34.4	-	-	-	31.1	31.6	30.6	-	-
	TransE	90.1	90.2	88.9	91.6	69.9	34.3	87.6	88.5	82.3	95.6	48.5	18.7
	RotatE	91.4	92.0	87.9	96.4	78.6	56.3	88.6	89.2	84.3	94.5	87.2	46.1
	ComplEx	81.4	81.4	81.4	81.3	52.7	43.7	80.8	81.1	80.4	81.9	65.2	29.2
	DistMult	83.2	83.2	83.4	82.9	45.0	44.9	88.4	88.2	89.8	88.6	75.3	36.3
	BoxE	**94.2**	94.2	94.1	94.2	78.7	32.8	93.0	93.5	92.8	94.2	**91.3**	53.2
	R-GCN	84.3	84.3	83.7	84.8	56.2	35.7	96.9	96.9	**99.7**	94.2	87.9	65.6
	AnyBURL	93.9	**94.3**	90.0	**98.9**	**80.8**	**56.9**	**99.8**	**99.9**	**99.7**	**100.0**	90.3	**67.2**
	RuleN	86.0	84.2	**96.3**	74.8	62.9	41.9	85.8	83.5	99.6	71.9	73.1	42.0
hier	SimpBL	-	36.1	39.5	33.2	-	-	-	31.8	35.7	28.6	-	-
	TransE	90.1	90.4	87.9	93.0	29.6	17.2	99.4	99.4	99.3	99.5	33.4	27.8
	RotatE	91.9	92.1	89.3	95.1	42.7	26.5	98.8	98.8	98.4	99.2	66.4	69.1
	ComplEx	81.3	80.5	84.0	77.3	26.4	16.7	81.0	80.1	84.0	76.5	53.3	16.5
	DistMult	80.5	80.9	79.5	82.3	27.0	19.7	83.6	84.1	81.8	86.4	59.6	28.5
	BoxE	**94.9**	**94.9**	94.2	95.7	77.2	32.9	**99.9**	**99.9**	**99.9**	99.9	86.7	57.6
	R-GCN	84.5	85.6	79.9	92.2	72.5	34.5	65.5	67.9	63.4	73.2	**95.5**	61.1
	AnyBURL	91.8	92.4	86.0	**99.9**	**82.1**	**56.2**	**99.9**	**99.9**	**99.9**	**100.0**	94.4	**73.5**
	RuleN	85.3	83.4	**96.4**	73.4	60.6	39.1	82.5	79.0	99.0	65.7	67.7	35.8
comp	SimpBL	-	25.8	30.9	22.2	-	-	-	31.6	29.8	33.6	-	-
	TransE	86.9	87.1	79.9	95.7	13.9	8.2	**99.7**	**99.7**	**99.7**	99.7	32.4	49.9
	RotatE	88.1	89.0	83.5	95.2	24.6	16.3	99.0	99.0	99.4	98.6	45.2	84.2
	ComplEx	68.6	70.9	66.1	76.5	9.7	7.3	84.5	85.1	81.7	88.9	45.2	9.5
	DistMult	70.4	72.6	67.6	78.4	9.8	8.2	95.4	95.3	98.2	92.5	50.9	64.7
	BoxE	90.1	90.6	87.0	94.5	39.3	23.7	**99.7**	**99.7**	**99.7**	99.7	81.3	61.9
	R-GCN	73.6	74.9	71.3	79.0	8.2	9.8	75.2	76.6	72.6	81.1	41.9	9.0
	AnyBURL	**91.7**	**92.3**	86.3	**99.2**	**39.4**	**34.1**	99.3	99.3	98.7	**99.9**	**92.9**	**93.4**
	RuleN	86.5	85.1	**94.3**	77.6	31.7	26.1	91.7	91.0	98.7	84.5	75.7	69.0

Table 5.4: Evaluation results on LogInfer-FB$^Y_{\text{qg}}$ and LogInfer-WN$^Y_{\text{qg}}$ (in %) with $Y \in \{\text{inter}, \text{trian}, \text{diam}\}$.

		LogInfer-FB$^Y_{\text{qg}}$						LogInfer-WN$^Y_{\text{qg}}$					
Y	Model	AUC	F1	Prec.	Rec.	R-M.	C-M.	AUC	F1	Prec.	Rec.	R-M.	C-M.
inter	SimpBL	-	65.5	**90.8**	51.2	-	-	-	-	-	-	-	-
	TransE	87.1	87.3	86.2	88.4	20.9	44.6	-	-	-	-	-	-
	RotatE	**89.3**	**89.4**	89.1	89.6	33.4	**78.4**	-	-	-	-	-	-
	ComplEx	83.4	82.0	89.3	75.8	17.7	65.1	-	-	-	-	-	-
	DistMult	85.7	86.1	83.6	88.7	20.3	63.2	-	-	-	-	-	-
	BoxE	85.3	88.9	85.3	**95.1**	60.7	40.5	-	-	-	-	-	-
	R-GCN	85.9	85.3	89.0	81.9	61.1	44.2	-	-	-	-	-	-
	AnyBURL	88.2	88.4	87.5	89.3	**76.4**	71.0	-	-	-	-	-	-
	RuleN	80.2	78.1	87.5	70.5	58.9	52.8	-	-	-	-	-	-
trian	SimpBL	-	52.2	49.4	55.2	-	-	-	56.3	87.7	41.5	-	-
	TransE	91.7	92.1	88.2	96.3	23.5	18.9	80.2	79.2	88.9	71.4	36.7	48.8
	RotatE	**93.1**	**93.1**	**90.8**	95.6	31.0	**31.5**	89.1	88.5	**93.4**	84.1	43.0	48.2
	ComplEx	77.0	77.9	74.9	81.2	23.5	18.9	74.4	70.8	82.2	62.2	27.1	12.6
	DistMult	81.2	81.9	78.7	85.4	20.3	25.4	83.8	82.3	90.3	75.6	30.7	31.8
	BoxE	82.7	85.0	75.0	**98.0**	**36.3**	21.1	58.9	70.0	54.9	**96.3**	69.5	55.5
	R-GCN	85.3	87.0	78.6	97.3	35.1	22.5	86.6	85.7	91.9	80.2	75.8	71.3
	AnyBURL	86.4	87.1	83.3	91.2	33.5	26.6	**89.7**	**89.8**	89.2	90.4	**86.3**	**81.7**
	RuleN	84.8	84.5	86.6	82.5	27.4	19.8	82.5	80.8	89.3	73.9	79.5	64.1
diam	SimpBL	-	60.2	67.6	54.2	-	-	-	48.7	69.3	37.5	-	-
	TransE	77.9	80.4	72.1	91.0	20.3	13.8	52.2	67.5	51.1	99.5	23.2	26.8
	RotatE	83.4	84.9	77.9	93.2	25.5	21.4	77.5	80.2	71.8	90.8	53.4	71.6
	ComplEx	64.2	70.3	60.1	84.8	8.4	6.2	55.0	67.0	52.8	91.6	35.7	1.6
	DistMult	77.6	79.6	73.0	87.4	11.9	15.5	69.3	74.4	63.8	89.1	45.6	23.4
	BoxE	77.5	80.5	71.2	92.5	17.3	16.8	72.8	76.5	67.5	88.2	80.5	58.8
	R-GCN	**86.6**	**87.3**	**83.1**	92.0	20.5	18.2	**94.5**	**94.4**	**95.3**	**93.5**	**89.6**	**70.9**
	AnyBURL	67.8	73.3	57.9	**99.9**	**39.5**	**35.2**	73.8	77.2	63.9	97.5	87.3	61.5
	RuleN	69.6	71.8	62.1	84.9	30.2	24.1	61.4	66.9	55.5	84.5	61.8	44.7

Table 5.5: Evaluation results on LogInfer-LUBM$^{Y}_{\text{qg}}$ (in %)

		LogInfer-LUBM$^{Y}_{\text{qg}}$					
Y	Model	AUC	F1	Prec.	Rec.	R-M.	C-M.
all	SimpBL	-	64.3	49.2	92.8	-	-
	TransE	96.3	96.3	95.6	97.0	75.5	26.8
	RotatE	96.3	96.5	95.8	97.3	96.7	49.4
	ComplEx	89.7	89.4	93.4	85.8	84.2	58.9
	DistMult	92.7	92.4	95.9	89.1	86.9	**62.5**
	BoxE	97.4	97.4	95.7	99.3	**100.0**	12.5
	R-GCN	94.1	94.1	94.1	94.0	82.7	34.9
	AnyBURL	**98.1**	**98.4**	**97.0**	**99.9**	**100.0**	22.0
	RuleN	88.3	91.2	95.0	87.6	89.0	18.1
no_type	SimpBL	-	62.4	48.2	88.8	-	-
	TransE	98.2	98.3	96.7	**99.9**	97.2	45.7
	RotatE	98.5	98.4	97.1	99.8	97.6	55.1
	ComplEx	89.9	89.3	94.6	84.6	86.3	49.7
	DistMult	93.3	93.1	95.4	90.9	84.5	63.2
	BoxE	99.3	99.3	98.8	99.8	**100.0**	10.1
	R-GCN	94.7	95.2	93.9	96.6	81.5	33.2
	AnyBURL	**99.9**	**99.9**	**100.0**	99.8	**100.0**	83.5
	RuleN	92.4	96.5	94.9	98.2	98.5	**98.2**

(90.9% on average for symmetry and 96% for intersection); this aligns with previous criticism indicating that such negative examples are trivial to recognise. In contrast, performance consistently degraded as we adopted increasingly fine-grained methods: average scores for relevance-aware corruption drop to 88% (symmetry) and 91.7% (intersection), and further degrade to 85.5% and 89.3% respectively for position-aware corruption. In turn, average performance drops to 85.6% for the intersection pattern for the query-guided approach.

These results support the effectiveness of our proposed methods in generating challenging negative examples. Overall, our findings indicate that classification performance heavily relies on the choice of negative examples, and thus highlights the importance of devising carefully designed methods for this choice in KG completion benchmarks.

5.5.4 Analysis of Extracted Rules

AnyBURL and RuleN explicitly construct sets of Datalog rules; therefore, we can compare the sets of rules returned by these systems with those in the benchmarks, $\mathcal{R}$, without relying on specific test examples. To this end, we considered LogInfer-FB$^{Y}_{Z}$

Table 5.6: AUC scores with various negative example generation methods for classification-based evaluation (in %)

	LogInfer-FB$^{\text{sym}}_Z$			LogInfer-FB$^{\text{inter}}_Z$			
Z	rc	rb	pa	rc	rb	pa	qg
TransE	62.6	56.7	47.0	**98.9**	93.9	94.4	87.1
RotatE	95.7	91.0	86.8	98.7	**95.5**	93.8	**89.3**
ComplEx	79.7	78.5	73.0	88.8	86.9	81.1	83.4
DistMult	**99.8**	**99.4**	97.5	97.0	92.9	84.9	85.7
BoxE	94.0	89.1	92.8	97.5	95.1	**95.9**	85.3
R-GCN	98.4	96.8	92.8	96.6	93.1	87.4	85.9
AnyBURL	98.1	96.7	96.0	97.5	91.4	90.8	88.2
RuleN	98.6	95.8	**98.0**	92.8	84.7	85.8	80.2

for all patterns Y as representative benchmarks (note that the value of Z is irrelevant since rule-based systems do not exploit negative training examples), and computed the following metrics:

1. the percentage ϵ_{ent} of benchmark rules $\mathcal{R}$ logically entailed by the rule sets generated by the systems; and

2. the percentage ϵ_{cont} of rules in $\mathcal{R}$ syntactically generated (and therefore also entailed) by the systems up to variable renaming and permutations of body atoms.

It is worth emphasising that these systems generate very large rule sets because rules are extracted by identifying dependencies in the training data, which includes the original KG; as a result, many generated rules are irrelevant to $\mathcal{R}$. Therefore, computing the percentages of extracted rules that are entailed by the benchmark rules is not very meaningful.

The results are summarised in Table 5.7. Over 95% benchmark rules corresponding to simple patterns are syntactically included in the output rule sets of AnyBURL and RuleN; this aligns with their superior performance on these patterns.

We further analysed rules for complex patterns. AnyBURL and RuleN only generate Datalog rules of certain syntactic form; for instance, AnyBURL and RuleN cannot syntactically output rules for intersection, triangle, or diamond patterns. A very large proportion of these rules are, however, entailed by the systems' output rule sets; this can be explained by the fact that the systems are generating simpler rules instead. To verify this, we have focused on the triangle pattern and have generated all the rules $\mathcal{R}^-$ obtained by selecting subsets of body atoms in the benchmark rules

Table 5.7: Results for logical entailment on LogInfer-FB^Y_Z (in %)

	Y	sym	inver	hier	comp	inter	trian	diam
ϵ_{ent}	AnyBURL	**100.0**	**98.5**	**99.0**	**95.0**	90.5	**99.5**	**89.5**
	RuleN	98.0	93.5	**99.0**	92.0	**97.5**	94.0	73.5
ϵ_{cont}	AnyBURL	96.0	**96.5**	**95.5**	**34.0**	0.0	0.0	0.0
	RuleN	**98.0**	70.5	69.5	23.0	0.0	0.0	0.0

for the pattern as described in Section 5.3.3. We could verify that 99.5% of these rules were entailed by the rules extracted by AnyBURL, and 89.5% of them were entailed by the rules extracted by RuleN. This explains the performance drop observed for rule-based systems on complex patterns and further supports the identified need for better negative example generation methods that penalise systems accordingly.

5.6 Summary

In this chapter, we comprehensively investigated and analysed existing KG completion benchmarks. A key feature of ML approaches for KG completion is their ability to learn inference patterns, so that the predicted facts are the results of applying these patterns to the KG. Standard completion benchmarks, however, are not well-suited for evaluating models' abilities to learn patterns, because the training and test sets of these benchmarks are a random split of a given KG and hence do not capture the causality of inference patterns. In this chapter, we have presented a novel approach for generating challenging inferential KG completion benchmarks. On the one hand, our approach ensures that models are exposed during training to both premise and conclusion triples for selected rules, and that triples in the test set have supporting evidence in the training set. On the other hand, our methods for generating negative examples ensure that models are penalised for learning unintended rules and yield examples that are both relevant and challenging to classify.

Overall, our findings highlight the gaps between theoretical and empirical results concerning models' ability to capture inference patterns. In spite of the significant contributions of our approach, it is imperative to consider some constraints that warrant further studies. First of all, as discussed in Section 5.3.1, our approach generates rules with randomness, and it is likely that the rules do not make sense in the semantic perspective. Similarly, as we generate negative examples based on the data and logic entailment, there is not a guarantee that the generated examples are negative in the semantic sense. Suppose there is only one rule (5.1) that a model aims to learn, and

our negative examples will penalise the models learning the rule (5.4) since rule (5.4) logically entails rule (5.1), but not the other way round. However, rule (5.4) actually holds in the semantic sense. A possible solutions is to check carefully the rules contributing to this process, or we can incorporate semantic information to the generation of the benchmarks. Moreover, in this work, we focus on the transductive KG completion problem. However, it would also be interesting to extend to the scenario of inductive KG completion, where the testing triples may involve constants unseen during training, and to see how the inductive KG completion systems perform on such inferential benchmarks. In addition, as our benchmarking approach is flexible and allows generating benchmarks for arbitrary inference patterns, it would also be interesting to explore more complex patterns, such as those with negations. Another interesting direction would be to leverage the benchmarks to further investigate the explainability of the KG completion systems. For instance, since we are able to identify the premise and conclusion triples of a rule in the KG, we can remove one premise triple from the training set and check if the model still predicts the corresponding conclusion triple. If not, we can draw a conclusion that the model made the prediction based on this specific rule. We believe that our work opens the door to these valuable and meaningful research problems for future investigation.

Chapter 6

Conclusions

We conclude the thesis with this chapter, where we first summarise our contributions in Section 6.1, and then discuss potential research directions opened by this thesis in Section 6.2.

6.1 Summary

With the advancement of the Semantic Web, KGs are reshaping information representation, organisation, and management paradigm. KGs describe the relations between entities using the graph structure, and play a significant role in propelling AI research. On the one hand, researchers have dedicated great efforts in investigating how to construct and improve the quality of KGs. We consider KG completion, a task of extending a KG with missing links that are supposed to hold, as a focal point in this context. One of the most common solutions for this task relies on the graph embedding techniques, which have achieved promising results for the transductive setting. However, they are not applicable to the inductive setting, where missing triples may mention constants unseen during training. On the other hand, there is also an increasing interest in a wide range of applications for KGs, where we choose to delve into the task of recommender systems, which aim at suggesting relevant items to users. KGs have shown great success in improving the accuracy, diversity, and interpretability of the recommendation results. Nevertheless, few of the knowledge-enhanced systems are inductive, in other words, able to make predictions in the situation where new objects are added, such as new users, items, or KG constants.

Deep learning has dramatically boosted the performance of tasks in fields such as CV and NLP. In recent years, GNNs have emerged as a prominent class of deep

learning approaches applicable to the graph domain, and have achieved superior performance in a variety of tasks on graph structured datasets including KGs. While there are plenty of research endeavours for leveraging GNNs to deal with KG completion and knowledge-enhanced recommendation, there still remained several open challenges. First of all, the GNN-based inductive KG completion systems often require pre-defined scoring functions to make predictions, and sometimes suffer from the scalability issue. Moreover, existing recommender systems are still limited, with none simultaneously supporting both the inductive setting and KGs. Finally, although previous research has analysed the KG completion systems' capability to capture inference patterns from a theoretical perspective, it still remained unexplored for assessing them from an empirical view.

In this thesis, we aimed to address the aforementioned challenges, propose powerful GNN-based approaches for inductive KG completion and inductive knowledge-enhanced recommendation, better understand what the models have learnt, and unravel the insights embedded in the deep learning approaches. We highlight our contributions in this thesis as follows.

In Chapter 3, we proposed a GNN-based approach for inductive KG completion, where our novelty mainly lies in the pair-wise encoding. Meanwhile, we implemented a system, INDIGO, based on this approach. We first analysed the efficiency of our approach by a theoretical analysis showing that the pair-wise encoding only resulted in a linear growth in the graph size with an theoretical analysis, and empirical experiments showing that INDIGO can be trained and tested within the shorter time compared with the baselines. Moreover, we also elaborated on the rationale behind our contribution through a discussion about unary and binary encoders. Then, in our experiments, we demonstrated the effectiveness of INDIGO by comparing it with state-of-the-art systems on a series of inductive benchmarks as well as a new benchmark proposed in our work. The results showed that INDIGO outperformed the other competitors with a large margin. Moreover, we presented an ablation study of GNN variants and showed that GCN was more effective in practice. Finally, we also validated the INDIGO's capability to capture inference patterns represented in logic rules.

In Chapter 4, we further applied the neural architecture proposed in Chapter 3 to the inductive knowledge-enhanced recommendation task, and implemented a recommender system system InKER. We showed with a comprehensive literature review that InKER was the first to support both inductive settings and external KGs. Furthermore, empirically, INKER demonstrated superior performance compared with

state-of-the-art baselines. Finally, in our experiments, we also investigated the impact of different GNN structures on the system performance, and examined the practical implications of incorporating KGs when the rating matrix becomes sparse.

In Chapter 5, we moved back to the KG completion problem, and aimed at revisiting the benchmarks used in this domain and understanding the inference patterns that were learnt by the models. In particular, we start with the transductive settings in this chapter, which is more straightforward and there are more existing benchmarks to analyse and compare. In the future, we will also consider the inductive settings. We proposed theoretical requirements for inferential KG completion benchmarks, and showed that none of the existing benchmarks can satisfy all the requirements. Based on the analysis, we proposed a novel approach enabling researchers to create inferential benchmarks and assess the performance of their own KG completion systems. Notably, we presented three interesting sampling strategies to generate challenging negative examples for the benchmarks. We constructed a suite of 37 benchmarks based on our approach, and evaluated the representative KG completion systems on the benchmarks. Our empirical findings highlighted the effectiveness of our approach, and some systems' remarkable performance in spite of their inability to capture certain patterns in theory.

Altogether, we believe this thesis contributes to the research progress towards deep learning with KGs using GNNs, where we delved particularly into two interesting tasks, inductive KG completion and inductive knowledge-enhanced recommendation. In addition to boosting the predictive performance on these tasks, the thesis also engaged in-depth reflections on the KG completion benchmarks and the models' ability to make inference. We believe this has, to some extent, alleviated the limitations of the black-box nature of deep learning approaches, facilitating the progress towards strong AI.

6.2 Future Work

Through this thesis, we have embarked on a journey towards inductive KG completion, inductive knowledge-enhanced recommendation, and inferential KG completion benchmarks. In this section, we highlight several future directions that are worth exploring.

One straightforward direction is to dive deeper into the innovative pair-wise encoding method proposed in this thesis. First of all, even though we have discussed the insights behind the pair-wise encoder in Section 3.4, it would be interesting to

conduct a formal logic characterisation on INDIGO. Moreover, one can potentially apply this encoding method to other tasks, such as social recommendation, or further combine this method with other advanced techniques. Inspired from k-GNN [94], it would also be interesting to try to increase the number of nodes as a unit in the encoding, such as triple or quadruple, and to further explore the advantages and disadvantages of such encoding methods. We refer the read to a recent work for link prediction with relational hypergraphs [59].

Another interesting direction is to consider a more general inductive setting for the tasks. Concretely speaking, this thesis only considers the setting where the missing triples mention *constants* unseen during training. However, it is also likely that the missing triples mention *relations* or *types* unseen during training, for which our approaches are not applicable. Therefore, we highlight here the significance of devising approaches applicable to this setting. There have been research efforts towards this direction [45, 106, 121, 172], and they have made some progress in advancing the performance in this task. It would also be interesting to see if our approach can be extended to this case, and even see if there can be any techniques supporting making predictions for triples involving both unseen constants and unseen relations.

Moreover, since LLMs, especially ChatGPT [80], have shown incredibly powerful performance and made profound influence in redefining the research paradigm of natural language processing, it is also worth exploring towards the application of LLMs on the KG completion and recommendation tasks. Despite there have already been attempts of using LLMs [34, 37, 136, 168] for these tasks, they often focus on utilising the text description of the KG constants, which is not always available. However, it still awaits more exploration of leveraging LLMs to capture the graph structure information in either the KG or the user-item interaction graph. A key challenge lies in the transformation of the graphs, as the LLMs can only take sequential data as input. Hence, it would be important to find feasible strategies converting graphs to sequences while preserving the expressive structure information as much as possible.

Our work in Chapter 5 also opens the door to a variety of future investigations. First and foremost, it would be intuitive and interesting to extend our setting to inductive KG completion, and assess the ability of inductive KG completion systems, including our INDIGO, to capture inference patterns. It is worth mentioning that careful designs of splitting and negative sampling are needed for this case. Second, as our proposed approach is not limited to the inference patterns listed in our work but allows for generating benchmarks based on arbitrary patterns, one promising direction is to explore more sophisticated patterns such as patterns with negations

or specific constants. Furthermore, another compelling direction is to exploit the inferential benchmarks for a in-depth investigation of the explainability of these KG completion systems. One limitation of these benchmarks is that we can only draw a conclusion that one system demonstrates good or bad performance on a benchmark with a collection of rules, but can not say if the system makes predictions because of the rules. As a consequence, we can remove one premise triple from the training set and check if the model still predicts the corresponding conclusion triple. If not, we can say that the predictions are made based on this specific rule. Last but not least, as our findings in Section 5 highlighted the gaps between theoretical and empirical results concerning models' ability to capture inference patterns, it would also be interesting to conduct a comprehensive analysis about the causes of these gaps. We believe this analysis can also be helpful for improving the existing approaches or proposing new approaches.

Finally, it would be interesting to extend our works in Chapter 3, Chapter 4, and Chapter 5 to the temporal KGs, which consist of time-specific knowledge. This aligns with the practice in the real world, as many facts in the KGs are time-specific and only valid for a certain period of time. When applying the normal approaches to the temporal case, some open challenges arise such as how to effectively incorporate the time information and how to dynamically capture the graph structure information. One potential direction is to combine GNNs with neural networks designed for sequential data such as RNNs, but it still requires careful designs when combining the two types of neural networks. Another interesting research point pertains to the negative sampling strategies: by incorporating temporal information, the number of negative candidates will increase sharply and it becomes more difficult to filter out the ones that are trivial to identify and generate challenging negative examples. We refer the readers to a recent survey about this topic [143].

Deep learning has vast potential. Although recent advanced deep learning techniques, such large pre-trained models stand out as a game changer of AI research, highlighting the significance of large-scale training data and computing power, there is still a long way to go before achieving strong AI. One inherent limitation of these models is that they may lack real-world knowledge, and can provide erroneous information about facts in the world. Moreover, they are limited in making inferences, and may not be able to exhibit sophisticated reasoning or deep understanding like humans. KGs can serve as a good complement in this case, allowing the models to access more accurate and up-to-date knowledge, and enhancing the models' ability to provide logical and contextually appropriate predictions. Moreover, GNNs are a

natural and powerful tool for deep learning with KGs, and we can observe that it still remains a substantial challenge to improve the message-passing paradigm of GNNs in a deeper, more expressive and more scalable fashion. Therefore, we believe deep learning with KGs using GNNs will continue being a very valuable research direction, and they can potentially provide a new wave of AI techniques that are both powerful and explainable.

Bibliography

[1] Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A box embedding model for knowledge base completion. In *NeurIPS*, 2020.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016.

[4] Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In *SIGMOD*, pages 1995–2010, 2020.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, volume 4825, pages 722–735, 2007.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[7] Iti Bansal, Sudhanshu Tiwari, and Carlos R. Rivero. The impact of negative triple generation strategies and anomalies on knowledge graph completion. In *CIKM*, pages 45–54. ACM, 2020.

[8] John S Baras and George Theodorakopoulos. Path problems in networks. *Synthesis Lectures on Learning, Networks, and Algorithms*, 3(1):1, 2010.

[9] Pablo Barceló, Mikhail Galkin, Christopher Morris, and Miguel A. Romero Orth. Weisfeiler and leman go relational. In *LoG*, volume 198, page 46, 2022.

[10] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[11] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NeurIPS*, pages 4502–4510, 2016.

[12] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[13] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[14] Tim Berners-Lee. Semantic Web road map, 1998.

[15] Max Berrendorf, Evgeniy Faerman, Laurent Vermue, and Volker Tresp. Interpretable and fair comparison of link prediction or entity alignment methods with adjusted mean rank. *CoRR*, abs/2002.06914, 2020.

[16] Federico Bianchi, Gaetano Rossiello, Luca Costabello, Matteo Palmonari, and Pasquale Minervini. Knowledge graph embeddings and explainable AI. In *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*, volume 47 of *Studies on the Semantic Web*, pages 49–72. IOS Press, 2020.

[17] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[18] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[19] Guillaume Bouchard, Sameer Singh, and Théo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI Spring Symposia*, 2015.

[20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu,

Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[21] Albert Camus. *The Myth Of Sisyphus And Other Essays.* Penguin Classics, 1955.

[22] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, 2012.

[23] Yixin Cao, Xiang Ji, Xin Lv, Juanzi Li, Yonggang Wen, and Hanwang Zhang. Are missing links predictable? an inferential benchmark for knowledge graph completion. In *ACL*, pages 6855–6865, 2021.

[24] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *WWW*, pages 151–161, 2019.

[25] Zongsheng Cao, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Dual quaternion knowledge graph embeddings. In *AAAI*, pages 6894–6902, 2021.

[26] Zongsheng Cao, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Geometry interaction knowledge graph embeddings. In *AAAI*, pages 5521–5529, 2022.

[27] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445, 2020.

[28] Jiajun Chen, Huarui He, Feng Wu, and Jie Wang. Topology-aware correlations between relations for inductive link prediction in knowledge graphs. In *AAAI*, pages 6271–6278. AAAI Press, 2021.

[29] Jiaoyan Chen, Pan Hu, Ernesto Jiménez-Ruiz, Ole Magnus Holter, Denvar Antonyrajah, and Ian Horrocks. Owl2vec*: embedding of OWL ontologies. *Mach. Learn.*, 110(7):1813–1845, 2021.

[30] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, Xi Chen, and Erik Bryhn Myklebust. An assertion and alignment correction framework for large scale knowledge bases. *Semantic Web*, 14(1):29–53, 2023.

[31] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, volume 119, pages 1725–1735, 2020.

[32] 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. In *IJCAI*, pages 6574–6582, 2023.

[33] Zhe Chen, Yuehan Wang, Bin Zhao, Jing Cheng, Xin Zhao, and Zongtao Duan. Knowledge graph completion: A review. *IEEE Access*, 8:192435–192456, 2020.

[34] Daniel Daza, Michael Cochez, and Paul Groth. Inductive entity representations from text via link prediction. In *WWW*, pages 798–808, 2021.

[35] Filipe de Sá Mesquita, Matteo Cannaviccio, Jordan Schmidek, Paramita Mirza, and Denilson Barbosa. Knowledgenet: A benchmark dataset for knowledge base population. In *EMNLP-IJCNLP*, pages 749–758, 2019.

[36] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, pages 1811–1818. AAAI Press, 2018.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. Association for Computational Linguistics, 2019.

[38] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *KDD*, pages 601–610, 2014.

[39] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, pages 2224–2232, 2015.

[40] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In *SEMANTiCS*, volume 1695, 2016.

[41] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *SIGKDD*, pages 1156–1165, 2014.

[42] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, pages 417–426. ACM, 2019.

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

[44] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *NeurIPS*, pages 6530–6539, 2017.

[45] Yuxia Geng, Jiaoyan Chen, Jeff Z. Pan, Mingyang Chen, Song Jiang, Wen Zhang, and Huajun Chen. Relational message passing for fully inductive knowledge graph completion. In *ICDE*, pages 1221–1233, 2023.

[46] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, volume 2, pages 729–734, 2005.

[47] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

[48] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. In *IJCAI*, pages 1725–1731, 2017.

[49] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semant.*, 3(2-3):158–182, 2005.

[50] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. In *IJCAI*, pages 1802–1808. ijcai.org, 2017.

[51] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[52] Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. Inductive link prediction for nodes having only attribute information. In *IJCAI*, pages 1209–1215, 2020.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[54] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, pages 639–648, 2020.

[55] David Herron, Ernesto Jiménez-Ruiz, and Tillman Weyde. On the benefits of owl-based knowledge graphs for neural-symbolic systems. In *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning*, volume 3432, pages 327–335, 2023.

[56] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, Sebastian Rudolph, et al. Owl 2 web ontology language primer. *W3C recommendation*, 27(1):123, 2009.

[57] 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.

[58] Yedid Hoshen. VAIN: attentional multi-agent predictive modeling. In *NeurIPS*, 2017.

[59] Xingyue Huang, Miguel Romero Orth, Pablo Barceló, Michael M Bronstein, and İsmail İlkan Ceylan. Link prediction with relational hypergraphs. *arXiv preprint arXiv:2402.04062*, 2024.

[60] Xingyue Huang, Miguel Romero, İsmail İlkan Ceylan, and Pablo Barceló. A theory of link prediction via relational weisfeiler-leman on knowledge graphs. In *NeurIPS*, 2023.

[61] Zan Huang, Xin Li, and Hsinchun Chen. Link prediction approach to collaborative filtering. In *JCDL*, pages 141–142. ACM, 2005.

[62] Nicolas Hubert, Pierre Monnin, and Heiko Paulheim. Beyond transduction: A survey on inductive, few shot, and zero shot link prediction in knowledge graphs. *arXiv preprint arXiv:2312.04997*, 2023.

[63] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, pages 687–696, 2015.

[64] Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, pages 381–388, 2006.

[65] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, pages 6348–6358, 2017.

[66] Bosung Kim, Taesuk Hong, Youngjoong Ko, and Jungyun Seo. Multi-task learning for knowledge graph completion with pre-trained language models. In *COLING*, pages 1737–1743, 2020.

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

[68] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

[69] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[71] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[72] Fritz Lehmann. *Semantic networks in artificial intelligence.* Elsevier, 1992.

[73] Cornelius T Leondes. *Expert systems: the technology of knowledge management and decision making for the 21st century.* Elsevier, 2001.

[74] Michelle M Li, Kexin Huang, and Marinka Zitnik. Graph representation learning in biomedicine and healthcare. *Nature Biomedical Engineering*, 6(12):1353–1369, 2022.

[75] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545, 2018.

[76] Ruifan Li, Hao Chen, Fangxiang Feng, Zhanyu Ma, Xiaojie Wang, and Eduard H. Hovy. Dual graph convolutional networks for aspect-based sentiment analysis. In *ACL*, pages 6319–6329, 2021.

[77] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015.

[78] Shuwen Liu, Bernardo Cuenca Grau, Ian Horrocks, and Egor V. Kostylev. INDIGO: gnn-based inductive knowledge graph completion using pair-wise encoding. In *NeurIPS*, pages 2034–2045, 2021.

[79] Shuwen Liu, Bernardo Cuenca Grau, Ian Horrocks, and Egor V. Kostylev. Revisiting inferential benchmarks for knowledge graph completion. In *KR*, pages 461–471, 2023.

[80] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, et al. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, page 100017, 2023.

[81] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *EMNLP*, pages 3219–3232, 2018.

[82] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. The Association for Computational Linguistics, 2015.

[83] Xinze Lyu, Guangyao Li, JiaCheng Huang, and Wei Hu. Rule-guided graph neural networks for recommender systems. In *ISWC*, 2020.

[84] Weizhi Ma, Min Zhang, Yue Cao, Woojeong Jin, Chenyang Wang, Yiqun Liu, Shaoping Ma, and Xiang Ren. Jointly learning explainable rules for recommendation with knowledge graph. In *WWW*, pages 1210–1221, 2019.

[85] Lester W. Mackey, Ameet Talwalkar, and Michael I. Jordan. Divide-and-conquer matrix factorization. In *NeurIPS*, pages 1134–1142, 2011.

[86] Sijie Mai, Shuangjia Zheng, Yuedong Yang, and Haifeng Hu. Communicative message passing for inductive relation reasoning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pages 4294–4302, 2021.

[87] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, 10 February 2004.

[88] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143, 2019.

[89] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In *ISWC*, volume 11136, pages 3–20, 2018.

[90] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[91] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *NAACL-HLT*, pages 777–782, 2013.

[92] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *NIPS*, pages 2204–2212, 2014.

[93] Changsung Moon, Paul Jones, and Nagiza F. Samatova. Learning entity type embeddings for knowledge graph completion. In *CIKM*, pages 2215–2218, 2017.

[94] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, pages 4602–4609, 2019.

[95] Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Maintenance of datalog materialisations revisited. *Artif. Intell.*, 269:76–136, 2019.

[96] Erik B. Myklebust, Ernesto Jiménez-Ruiz, Jiaoyan Chen, Raoul Wolf, and Knut Erik Tollefsen. Prediction of adverse biological effects of chemicals using knowledge graph embeddings. *Semantic Web*, 13(3):299–338, 2022.

[97] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *ACL*, pages 4710–4723, 2019.

[98] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfox: A highly-scalable RDF store. In *ISWC*, volume 9367, pages 3–20, 2015.

[99] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Q. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *NAACL-HLT*, pages 327–333, 2018.

[100] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816, 2011.

[101] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. Knowledge graph embeddings with node2vec for item recommendation. In *ESWC*, volume 11155, pages 117–120, 2018.

[102] Alexis Papadimitriou, Panagiotis Symeonidis, and Yannis Manolopoulos. Fast and accurate link prediction in social networking systems. *J. Syst. Softw.*, 85(9):2119–2132, 2012.

[103] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.

[104] Pouya Pezeshkpour, Yifan Tian, and Sameer Singh. Revisiting evaluation of knowledge base completion models. In *AKBC*, 2020.

[105] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3d graph neural networks for RGBD semantic segmentation. In *ICCV*, pages 5209–5218, 2017.

[106] Pengda Qin, Xin Wang, Wenhu Chen, Chunyun Zhang, Weiran Xu, and William Yang Wang. Generative adversarial zero-shot relational learning for knowledge graphs. In *AAAI*, pages 8673–8680, 2020.

[107] Yanru Qu, Ting Bai, Weinan Zhang, Jian-Yun Nie, and Jian Tang. An end-to-end neighborhood-based interaction model for knowledge-enhanced recommendation, 2019.

[108] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[109] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

[110] Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael M. Bronstein. Edge directionality improves learning on heterophilic graphs. *CoRR*, abs/2305.10498, 2023.

[111] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. DRUM: end-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, pages 15321–15331, 2019.

[112] Tara Safavi and Danai Koutra. Codex: A comprehensive knowledge graph completion benchmark. In *EMNLP*, pages 8328–8350, 2020.

[113] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, volume 307, pages 880–887, 2008.

[114] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[115] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, volume 10843, pages 593–607, 2018.

[116] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI*, pages 3060–3067, 2019.

[117] Tong Shen, Fu Zhang, and Jingwei Cheng. A comprehensive overview of knowledge graph completion. *Knowl. Based Syst.*, 255:109597, 2022.

[118] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. In *AAAI*, pages 1957–1964, 2018.

[119] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *IJCAI*, pages 1548–1554. ijcai.org, 2021.

[120] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, pages 926–934, 2013.

[121] Ran Song, Shizhu He, Suncong Zheng, Shengxiang Gao, Kang Liu, Zhengtao Yu, and Jun Zhao. Decoupling mixture-of-graphs: Unseen relational learning for knowledge graph completion by fusing ontology and textual experts. In *COLING*, pages 2237–2246, 2022.

[122] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.

[123] Chengcheng Sun, Chenhao Li, Xiang Lin, Tianji Zheng, Fanrong Meng, Xiaobin Rui, and Zhixiao Wang. Attention-based graph neural networks: a survey. *Artificial Intelligence Review*, pages 1–48, 2023.

[124] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.

[125] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha P. Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *ACL*, pages 5516–5522, 2020.

[126] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[127] Komal K. Teru, Etienne G. Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, volume 119, pages 9448–9457, 2020.

[128] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *CVSC*, pages 57–66, 2015.

[129] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48, pages 2071–2080. JMLR.org, 2016.

[130] Yi-Lin Tuan, Sajjad Beygi, Maryam Fazel-Zarandi, Qiaozi Gao, Alessandra Cervone, and William Yang Wang. Towards large-scale interpretable knowledge graph reasoning for dialogue systems. In *ACL*, pages 383–395, 2022.

[131] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion. *CoRR*, abs/1706.02263, 2017.

[132] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *ICLR*, 2020.

[133] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

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

[135] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[136] Bo Wang, Tao Shen, Guodong Long, Tianyi Zhou, Ying Wang, and Yi Chang. Structure-augmented text representation learning for efficient knowledge graph completion. In *WWW*, pages 1737–1748, 2021.

[137] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *CVPR*, pages 6450–6458, 2017.

[138] Hongwei Wang, Hongyu Ren, and Jure Leskovec. Relational message passing for knowledge graph completion. In *KDD*, pages 1697–1707. ACM, 2021.

[139] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM*, pages 417–426, 2018.

[140] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *KDD*, pages 968–977, 2019.

[141] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Multi-task feature learning for knowledge graph enhanced recommendation. In *WWW*, pages 2000–2010, 2019.

[142] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *WWW*, pages 3307–3313. ACM, 2019.

[143] Jiapu Wang, Boyue Wang, Meikang Qiu, Shirui Pan, Bo Xiong, Heng Liu, Linhao Luo, Tengfei Liu, Yongli Hu, Baocai Yin, et al. A survey on temporal knowledge graph completion: Taxonomy, progress, and prospects. *arXiv preprint arXiv:2308.02457*, 2023.

[144] Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. Simkgc: Simple contrastive knowledge graph completion with pre-trained language models. In *ACL*, pages 4281–4294, 2022.

[145] Peifeng Wang, Jialong Han, Chenliang Li, and Rong Pan. Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *AAAI*, pages 7152–7159, 2019.

[146] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Trans. Knowl. Data Eng.*, 29(12):2724–2743, 2017.

[147] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. In *SIGKDD*, pages 950–958, 2019.

[148] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. Learning intents behind interactions with knowledge graph for recommendation. In *WWW*, pages 878–887, 2021.

[149] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. AAAI Press, 2014.

[150] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*, pages 349–357, 2018.

[151] Donald A. Waterman. *A guide to expert systems*. Addison-Wesley, 1986.

[152] Yuze Wei, Jie Luo, and Huiyuan Xie. Kgrl: an owl2 rl reasoning system for large scale knowledge graph. In *12th International Conference on Semantics, Knowledge and Grids (SKG)*, pages 83–89, 2016.

[153] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. A neural influence diffusion model for social recommendation. In *SIGIR*, pages 235–244. ACM, 2019.

[154] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. In *WWW*, pages 2091–2102, 2019.

[155] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Junchi Yan, and Hongyuan Zha. Towards open-world recommendation: An inductive model-based collaborative filtering approach. In *ICML*, volume 139, pages 11329–11339, 2021.

[156] Guo-Sen Xie, Jie Liu, Huan Xiong, and Ling Shao. Scale-aware graph neural network for few-shot semantic segmentation. In *CVPR*, pages 5475–5484, 2021.

[157] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, pages 2659–2665, 2016.

[158] Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Representation learning of knowledge graphs with hierarchical types. In *IJCAI*, pages 2965–2971, 2016.

[159] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, pages 564–573, 2017.

[160] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. Graph contextualized self-attention network for session-based recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3940–3946, 2019.

[161] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[162] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *ICLR*, 2020.

[163] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, and Hanghang Tong. Dynamic knowledge graph alignment. In *AAAI*, pages 4564–4572, 2021.

[164] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

[165] Deqing Yang, Zengchun Song, Lvxin Xue, and Yanghua Xiao. A knowledge-enhanced recommendation model with attribute-level co-attention. In *SIGIR*, pages 1909–1912, 2020.

[166] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, pages 2319–2328, 2017.

[167] Yuhao Yang, Chao Huang, Lianghao Xia, and Chenliang Li. Knowledge graph contrastive learning for recommendation. In *SIGIR*, pages 1434–1443, 2022.

[168] Liang Yao, Chengsheng Mao, and Yuan Luo. KG-BERT: BERT for knowledge graph completion. *CoRR*, abs/1909.03193, 2019.

[169] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *NeurIPS*, pages 28877–28888, 2021.

[170] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983, 2018.

[171] Wenhui Yu, Zixin Zhang, and Zheng Qin. Low-pass graph convolutional network for recommendation. In *AAAI*, pages 8954–8961, 2022.

[172] Hanwen Zha, Zhiyu Chen, and Xifeng Yan. Inductive relation prediction by BERT. In *AAAI*, pages 5923–5931, 2022.

[173] Chen Zhang, Qiuchi Li, and Dawei Song. Aspect-based sentiment classification with aspect-specific graph convolutional networks. In *EMNLP*, pages 4567–4577, 2019.

[174] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *SIGKDD*, pages 353–362, 2016.

[175] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. STAR-GCN: stacked and reconstructed graph convolutional networks for recommender systems. In *IJCAI*, pages 4264–4270, 2019.

[176] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, pages 5171–5181, 2018.

[177] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *ICLR*, 2020.

[178] 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 *NeurIPS*, pages 9061–9073, 2021.

[179] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *NeurIPS*, pages 2731–2741, 2019.

[180] Xueliang Zhao, Wei Wu, Can Xu, Chongyang Tao, Dongyan Zhao, and Rui Yan. Knowledge-grounded dialogue generation with pre-trained language models. In *EMNLP*, pages 3377–3390, 2020.

[181] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal A. C. Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, pages 29476–29490, 2021.