# ON DATA ENGINEERING AND KNOWLEDGE GRAPHS

✦✦✦

## A REINFORCEMENT LEARNING SYSTEM FOR KNOWLEDGE GRAPH REASONING

# MIGUEL BERMUDO BAYO

### UNIVERSITY OF SEVILLE, SPAIN

UNIVERSIDAD DE SEVILLA

JANUARY, 2024

*"I see now that the circumstances of one's birth are irrelevant. It is what you do with the gift of life that determines who you are."*

— Mewtwo

*A Blanca,*
*Manuel, Eloisa,*
*Inma, Dani, David, Fernando y Agustín*

# Contents

## I   Preface

## II   Background Information

# III Our Proposal

# IV   Final Remarks

# List of Figures

# List of Tables

# Acknowledgments

*"Appreciation is a wonderful thing. It makes what is excellent in others belong to us as well."*

— Voltaire

Although this chapter is one that contains no information about the topics found throughout the rest of the pages of this book, it finds itself dear and near to my heart, as my time as a student comes to its end, a new page on the book of my life draws near. I take this opportunity to be grateful for the amazing people I have found along the way.

First and foremost I extend my most sincerest gratitude to my directors Inma and Dani, the people who have been closest to the development of these works and have helped shape the words you are about to read. They are leaders by example and whom without I would have found myself lost on a multitude of occasions, all while having the utmost respect and understanding of everyone around them. And to David, our director, I say thank you for the trust and confidence you have instilled in me on many more than one occasion, even in times I did not trust in myself, your enthusiasm and true passion for research and advancement of the fields you work on is inspirational and I feel very lucky to be working alongside all of them.

To the people who have accompanied me on the daily throughout the days, my colleagues Agustin, Fernando, Pepe and Paula with whom I have shared our own small piece of the university halls where, not so long ago, we were students (some still are), I thank them for all the great times, coffee break conversations, debates and brainstorms we have shared throughout or time together. You are what gave me a reason to visit that room on the second floor of module F, I wish every single one of you the very best in your endeavors and that you accomplish whatever your brilliant minds decide to pursue next, although I have no doubts about your success.

To my friends, who kept me sane during these times, took my mind off of my duties when needed giving me a moment of respite, pushed me to continue when

I faltered and had the patience to stand by my side through all, I appreciate all you being there.

And my most heartfelt thanks go to the people closest to me, to those who pushed me every step of the way, who picked me up every time I fell down and who always are there when I need them, my parents, whose interest In my progress was only peaked by their interest in my wellbeing and their unwavering love and support. And finally, to Blanca, for an infinitely long list of reasons which I am unable to fit in these pages for fear this book would become a trilogy instead.

To all of you, thanks.

# Agradecimientos

*"El aprecio es algo maravilloso. Hace que lo que
es excelente en los demás nos pertenezca también a
nosotros."*

— Voltaire

Aunque este capítulo no contiene información sobre los temas que se encuentran en el resto de las páginas de este libro, me resulta muy querido y cercano, ya que mi tiempo como estudiante llega a su fin y se acerca una nueva página en el libro de mi vida. Aprovecho esta oportunidad para dar las gracias a las personas increíbles que he encontrado por el camino.

Ante todo, extiendo mi más sincera gratitud a mis directores Inma y Dani, las personas que han estado más cerca del desarrollo de estos trabajos y que han ayudado a dar forma a las palabras que estáis a punto de leer. Son líderes con el ejemplo y sin ellos me habría encontrado perdido en multitud de ocasiones, todo ello con el máximo respeto y comprensión hacia todos los que les rodean. Y a David, nuestro director, le doy las gracias por la confianza y seguridad que me ha infundido en más de una ocasión, incluso en momentos en los que no confiaba en mí mismo, su entusiasmo y verdadera pasión por la investigación y el avance de los campos en los que trabaja es inspirador y me siento muy afortunado de trabajar junto a todos ellos.

A las personas que me han acompañado en el día a día a lo largo de los días, mis compañeros Agustín, Fernando, Pepe y Paula con los que he compartido nuestro pequeño trozo de la universidad donde, no hace tanto, éramos estudiantes (algunos todavía lo son), les doy las gracias por todos los grandes momentos, conversaciones en el descanso del café, debates y brainstormings que hemos compartido a lo largo de nuestro tiempo juntos. Vosotros sois los que me habéis dado una razón para visitar esa habitación de la segunda planta del módulo F, os deseo a todos y cada uno de vosotros lo mejor en vuestros objetivos y que logréis lo que vuestras brillantes mentes decidan hacer a continuación, aunque no tengo ninguna duda de vuestro éxito.

A mis amigos, que me mantuvieron cuerdo durante estos tiempos, me distrajeron de mis obligaciones cuando lo necesité dándome un momento de respiro, me empujaron a continuar cuando flaqueé y tuvieron la paciencia de estar a mi lado en todo momento, os agradezco a todos que hayáis estado ahí.

Y mi más sincero agradecimiento a las personas más cercanas a mí, a los que me empujaron en cada paso del camino, a los que me levantaron cada vez que me caí y a los que siempre están ahí cuando los necesito, a mis padres, cuyo interés en mi progreso sólo se vio superado por su interés en mi bienestar y su amor y apoyo inquebrantables. Y por último, a Blanca, por una lista infinitamente larga de razones que no soy capaz de encajar en estas páginas por miedo a que este libro se convirtiera en una trilogía.

A todos, gracias.

# Abstract

*"Brevity is a great charm of eloquence."*

— *Marcus Tulius Cicero*

Knowledge Graphs have been at the forefront of domain information storage since their inception. These graphs can be used as the basis for a number of smart applications, such as question answering or product recommendations. However, they are generally built in an automated unsupervised way, which frequently leads to missing information, usually in the form of missing links between related entities in the original data source, and which have to be added a posteriori by completion techniques.

Knowledge Graph Completion seeks to find missing elements in a Knowledge Graph, usually edges representing some relation between two concepts. One possible way to do this is to find paths between two nodes that indicate the presence of a missing edge. This can be achieved through Reinforcement Learning, by training an agent that learns how to navigate through the graph, starting at a node with a missing edge and identifying what edge among the available ones at each step is more promising in order to reach the target of the missing edge.

While some approaches have been proposed to this effect, their reward functions only take into account whether the target node was reached or not, and only apply a single Reinforcement Learning algorithm. In this regard, we present a new family of reward functions based on node embeddings and structural distance, leveraging additional information related to semantic similarity and removing the need to reach the target node to obtain a measure of the benefits of an action.

We introduce SpaceRL an end-to-end Python framework designed for the generation of reinforcement learning (RL) agents, which can be used in knowledge graph completion and link discovery. The purpose of the generated agents is to help identify missing links in a knowledge graph by finding paths that implicitly connect two nodes, incidentally providing a reasoned explanation for the inferred new link.

The generation of such agents is a complex task, even more so for a non-expert user, and to the best of our knowledge there do not exist tools to provide that kind of support.

SpaceRL is meant to overcome these limitations by providing a flexible set of tools designed with a wide variety of customization options, in order to be flexible enough to adapt to different user needs. It also includes a variety of state-of-the-art RL algorithms and several embedding models that can be combined to optimize the agent's performance. Furthermore, SpaceRL offers different interfaces to make it available either locally (programmatically or via a GUI), or through an OpenAPI-compliant REST API.

# Resumen

*"Una síntesis vale por diez análisis."*

— Eugeni d'Ors

Los grafos de conocimiento han estado a la vanguardia del almacenamiento de información de dominio desde su creación. Estos grafos pueden servir de base para una serie de aplicaciones inteligentes, como la respuesta a preguntas o las recomendaciones de productos. Sin embargo, por lo general se construyen de forma automatizada y no supervisada, lo que a menudo da lugar a que falte información, normalmente en forma de enlaces que faltan entre entidades relacionadas en la fuente de datos original, y que tienen que añadirse a posteriori mediante técnicas de compleción.

La compleción de grafos de conocimiento trata de encontrar los elementos que faltan en un grafo de conocimiento, normalmente aristas que representan alguna relación entre dos conceptos. Una posible forma de hacerlo es encontrar caminos entre dos nodos que indiquen la presencia de una arista que falta. Esto puede lograrse mediante el Aprendizaje por Refuerzo, entrenando a un agente que aprenda a navegar por el grafo, comenzando en un nodo con una arista ausente e identificando qué arista de entre las disponibles en cada paso es más prometedora para alcanzar el objetivo de la arista ausente.

Aunque se han propuesto algunos enfoques en este sentido, sus funciones de recompensa sólo tienen en cuenta si se ha alcanzado o no el nodo objetivo, y sólo aplican un único algoritmo de Aprendizaje por Refuerzo. En este sentido, presentamos una nueva familia de funciones de recompensa basadas en la incrustación de nodos y la distancia estructural, aprovechando información adicional relacionada con la similitud semántica y eliminando la necesidad de alcanzar el nodo objetivo para obtener una medida de los beneficios de una acción

SpaceRL es un marco integral en Python diseñado para la generación de agentes de aprendizaje por refuerzo (RL), que pueden utilizarse para completar grafos de

conocimiento y descubrir enlaces. El objetivo de los agentes generados es ayudar a identificar los enlaces que faltan en un grafo de conocimiento encontrando caminos que conectan implícitamente dos nodos, proporcionando de paso una explicación razonada del nuevo enlace inferido. La generación de tales agentes es una tarea compleja, más aún para un usuario no experto, y hasta donde sabemos no existen herramientas que proporcionen ese tipo de ayuda.

SpaceRL pretende superar estas limitaciones proporcionando un conjunto flexible de herramientas diseñadas con una amplia variedad de opciones de personalización, con el fin de ser lo suficientemente flexible como para adaptarse a las diferentes necesidades de los usuarios. También incluye una variedad de algoritmos RL de última generación y varios modelos de incrustación que pueden combinarse para optimizar el rendimiento del agente. Además, SpaceRL ofrece diferentes interfaces para que esté disponible de forma local (mediante programación o a través de una GUI), o a través de una API REST compatible con OpenAPI.

# Part I

# Preface

# Chapter 1

# Introduction

*"Above all, don't fear difficult moments. The best comes from them."*

— *Rita Levi-Montalcini*

The purpose of this book is to present our work in Knowledge Graph reasoning. This chapter provides the required information for the reader to follow the topics discussed in this dissertation, it is structured in the following sections:

Section 1.1, contains the reasons why we found the works we introduce could contribute to the current state of these research fields; Section 1.2, holds our hypothesis and thesis; Section 1.3, focuses on the contributions made and what they present; Section 1.4, describes the structure of the rest of this book.

## 1.1    Research context

In the last decade, available information online has increased 60 000%, from a modest 2 zettabytes of data ($10^{12}$ gigabytes) up to an estimate of 120 by the end of 2023[51]. The sheer volume of being generated on a daily basis calls for a structured and networked storage solution made possible by Knowledge Graphs, whose rise to popularity was all but inevitable. These data structures hold information from multiple domains by using triples, two information nodes that represent concepts connected by an edge constituting a relationship between them, this relation can be either directional or linear meaning that a concept is related to another following that direction but not vice-versa, (e.g. parents and children are directional relations while siblings are linear). This form of representation obtained by adhering to these rules confers information in a graph-like structure containing a web of facts with a high degree of connectivity, offering complex reasoned chains of information in an effective way.
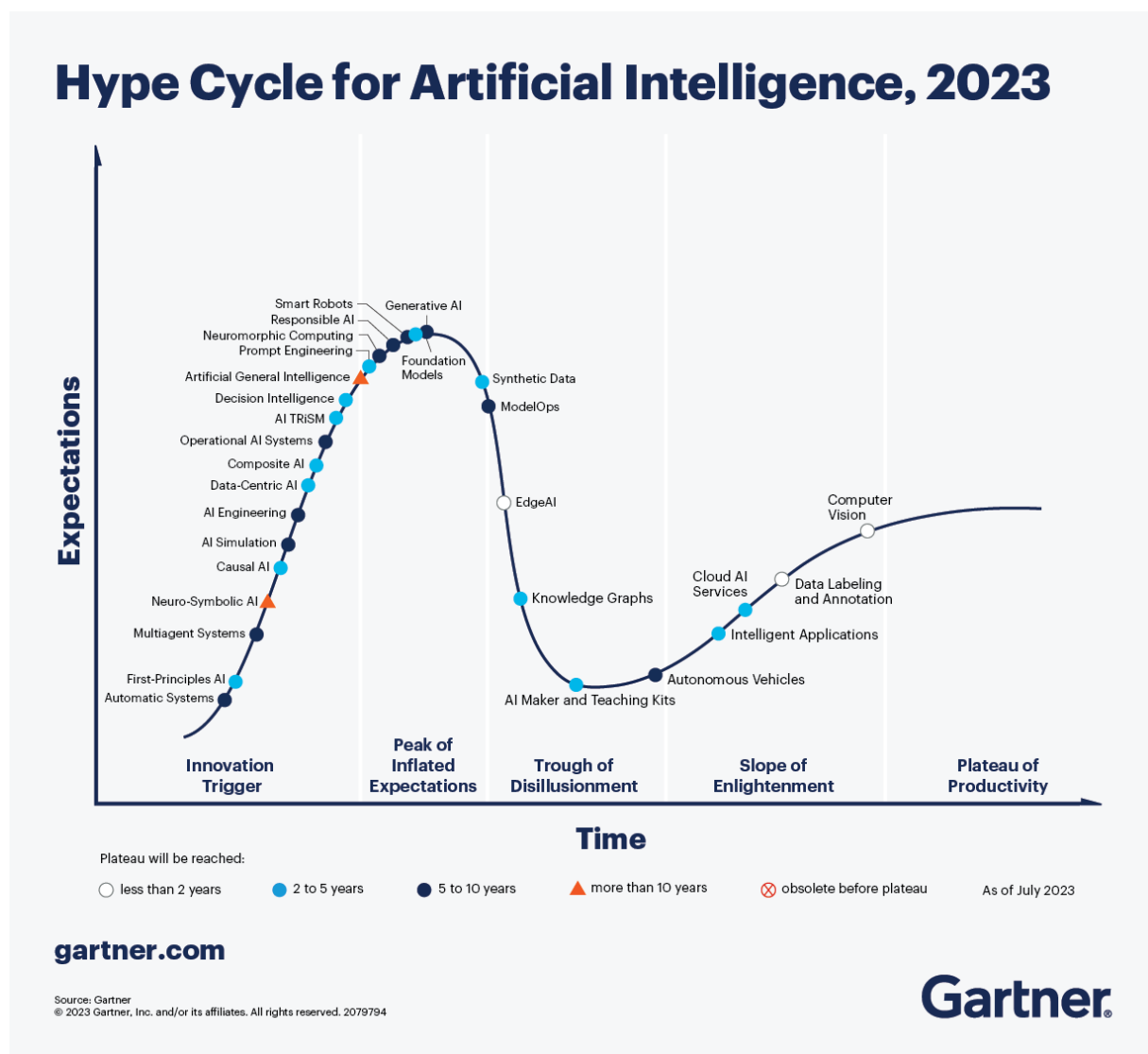


**Figure 1.1:** Gartner's 2023 hype chart for artificial intelligence.

The Gartner institution places Knowledge Graphs in the middle of their life cycle (cf. Figure 1.1) meaning that they can still benefit from active research and development and are still in their infancy in regard to their widespread applications in a multitude of fields. Some of the most notable fields and Knowledge Graphs corresponding to each of them are:

- **Enciclopedic** KGs compile factual knowledge facts or events sourced from different domains. Some examples are DBpedia[2] which compiles the knowledge found in Wikipedia articles, Freebase[4] which combines automatic processes from multiple sources as well as user contributions to wrangle up its data or YAGO[86] which adds a layer of complexity by adding temporal and geographical information to Wikipedia facts.
- **Linguistic** KGs compile facts about language and add a layer of information in the form of ontologies or external features on top. The most notable of these is WordNet[55] which provides hyponym and synonym relationships between words of the English language.
- **Enterprise** KGs support tech sector companies in their endeavors. Knowledge graphs were originally conceived for this purpose as the Google Knowledge Graph (GKG) [85] boasting an impressive 800 billion facts on 8 billion entities was incepted to aid in query answering. This graph compiles multi-domain information in order to rapidly respond to one of the many user queries made through their browser per day.

Knowledge Graph construction is generally an automatic task since the large volume of data they hold extends past the range of human processing abilities. Manually constructing a KG of large size becomes an exceptionally tedious, long, and costly process, or even downright impossible. However, there are some notable exceptions such as the high-quality datasets that were constructed through crowdfunding efforts such as Freebase[4] and Wikidata[102]. Automatic KG construction typically relies on semi-structured[44] information ranging from XML-type documents and HTML tables to plain text articles with a well-organized title structure. The data extraction process has evolved over time, from information extraction systems driven by designated rules or clustering [19, 111], to the current approaches such as entity recognition[33, 48], typing[67, 108] and linking[26, 42] or relation extraction and curation[113, 115].

These methods of automated construction allow for a massive volume of data to be processed, however, they present several limitations. First, the information they are built upon may be interpreted and linked erroneously or simply be demonstrably wrong [52], leading to incorrect facts being present in a KG. Second, Knowledge Graphs built from the same source might not be equal rendering them incompatible, they could represent the same facts with different nomenclature because they used a different schema to link that information [11], meaning that integrating Knowledge Graphs is

never a trivial task. Finally, Knowledge Graphs constructed in this manner will most likely be lacking information that was originally contained within the source material it used [5]. In addition, information sources generally do not include knowledge that the intended reader should already be aware of, in other words, no source will explicitly hold all information required for a single domain, therefore, Knowledge Graphs inherently lack triples that could otherwise exist in them. Some techniques aim to fill this information gap in automatically constructed Knowledge Graphs, known as KG completion, and their focus is on finding links between existing entities that are not present in the KG but should be. However, KG completion efforts have a limitation shared by multiple binary classifiers in the literature, they generally provide an answer to a query that can be accompanied by a degree of confidence in the form of a percentage. This lack of explainability of their answers is a limitation in KG completion that KG reasoning aims to complement by providing reasoned paths to accompany their responses. In this way, a provided answer can be human-friendly as well as serve the same purpose in regards to KG completion.

Existing literature displays the multiple ways in which KG completion and reasoning have been tackled. Completion efforts can be separated into one of four existing categories[76], they are as follows. **Semantic matching models**, compute a scoring function by measuring the semantic similarities of entity or relation embeddings in latent embedding space, they do so by applying **Neural Network** or **Tensor Factorization** models. NN models [81, 96] present a challenging problem as they require the effective encoding of world knowledge using powerful models that try to approximate human cognition by replicating the neural structures found in our brains. Research has shown that neural networks can capture the semantic features of entities and relations intelligently and model the semantic relationships between discrete entities, ultimately leading to more accurate embeddings of KGs. Tensors[3, 61, 81] and their decompositions are widely used in data mining and machine learning problems. their usefulness comes from the representation they offer as KG entities and triples can be represented as tensors, KG completion can be tackled as a binary tensor completion problem[76].

**Translational models**[6, 18, 46, 87, 97, 103] encode KG entities and relations as low-dimensional numerical vectors. a distance scoring function is then applied in the N-dimensional vectorial space reflecting the correctness of a proposed triple, then a ranking loss function is applied to learn the top translation relation between the entities in question. **Structural models**[54, 65] The inherent rich information inside KGs plays an important role in capturing useful features of knowledge embeddings for KG completion. the common internal information inside KGs includes node attribute information, entity-related information, relation-related information, neighborhood information, and relational path Information. This dissertation's proposal expands on relational path reasoning combined with translational elements as part of the

contribution.

Obtaining information from relational paths contained within a KG is usually performed by applying machine learning algorithms; path capture, path classification[107], path augmentation[31, 50] or multi-hop KG reasoning[13, 14, 45, 95]. Most of these approaches present an inherent flaw, they use a path ranking NN as a binary classifier which causes feature explosion for large KGs and lacks background information on the selected path, both of those shortcomings are solved by KG reasoning methods. That is why in our works we focused on the development of a multi-hop reasoning approach that adds explainability and prevents feature explosions for large KGs.

There are several options for the construction of artificial intelligence capable of performing KG reasoning, namely, supervised, unsupervised and reinforcement learning.

Supervised learning requires providing a golden truth in a labeled dataset, which means knowing the optimal outcome beforehand, this is useful for tasks that require classification and regression problems, predicting a value in continuous or discrete spaces. This is simply not possible for many problems in the field, in this case, it would mean obtaining the best path connecting two given nodes of a Knowledge Graph for the particular problem being solved. This would require expert knowledge and deep analysis for each of the nodes, a task which is simply unfeasible for sizeable Knowledge Graphs.

Unsupervised learning operates without a labeled dataset, aiming to discern underlying patterns within a problem and approximate the output. However, traditional unsupervised methods frequently encounter challenges in managing the intricacies of real-world knowledge graphs, particularly when confronted with ambiguous or missing information.

Reinforcement learning (RL) emerges as a promising paradigm to address these limitations. RL introduces an interactive learning framework where an agent learns to navigate and interact with the knowledge graph by using it as the environment of the problem. RL algorithms can dynamically adapt their strategies, effectively dealing with the uncertainties present in an automatically constructed KG even in the absence of human supervision.

In practice, Reinforcement Learning focuses on solving a stochastic gradient ascent problem in a KG reasoning scenario, that will be described in depth in chapter 5, it is a policy gradient method seeking to optimize the quality of the found path based on the metrics of some reward functions.

It does so by traversing through the graph nodes as states by using connected relations as the action space for that state. The objective of the Agent is to learn the

best path that answers a particular query. This query can be interpreted in the form of a question that needs answering, such as, "who is the partner of Keanu Reeves?" This, in the KG nomenclature, is expressed as a triple missing its end node (Keanu Reeves, is_partner_of, ?), the agent is then expected to find the destination node while training in order to generate new paths of information from the unanswered queries contained in the KG.

Several proposals have tackled this problem in the past [16, 45, 105, 107]. However they present several shortcomings, for instance, they require the embedding models to be computed before attempting the problem. Also, they restrict themselves to using classical RL backpropagation algorithms and overlook the application of more modern options such as Proximal Policy Optimization (PPO)[73] or Soft Actor-Critic (SAC) [29] while also relying on simple reward structures. Finally, most of these proposals are not distributed as usable tools intended for final users. Even if they make their implementation publicly available, their code is generally intended for the sake of reproducibility of their experimental results, and they often lack any degree of customization or flexibility, meaning they usually can only work on a number of predefined datasets as input.

In this dissertation work, we aim to combine the benefits from RL pathfinding with the power of representational embeddings to infer fairly long and explainable paths, useful for KG-based applications, and doing so with on-the-fly embedding generation and embedding-based rewards combined with step-based rewards.

The tool presented is highly configurable, allowing for reward calculation to be modified with a combination of several options, customizing the policy intermediate activation function and regularization and allowing the user to apply state-of-the-art RL algorithms out of the box, which improves performance and helps avoid reward plateaus while training.

Finally, we aim to provide a versatile tool intended for users with different levels of expertise, from novices to experts. It allows comprehensive and flexible customization for advanced users, who may prefer to install SpaceRL as a server for their local usage or to become a service provider for third parties. On the other hand, it also offers a simple MLaaS interface, intended for a more untrained end user. Machine Learning as a Service (MLaaS) has gained traction in recent years since it adds layers of abstraction that create a black-box simplified interface for a non-expert final user. SpaceRL offers RL model generation and usage as a service capability, either locally through its GUI or as a deployable REST API for third-party consumption. Therefore, it is, to the best of our knowledge, the first turnkey tool to provide such RL KG completion and reasoning functionalities.

## 1.2   Research rationale

In this section, we present the hypothesis that has motivated our research work in the context of Knowledge Graph reasoning and state our thesis, which we prove in the rest of the dissertation.

### 1.2.1   Hypothesis

Currently, Knowledge Graphs present a tantalizing alternative for structuring information about one or multiple domains. They offer a semantic connection between the data nodes which can be used for a multitude of applications such as question answering [13, 14], recommendation systems [105], biomedical studies [60] or natural language processing tasks. The nature of these knowledge graphs makes them inherently incomplete as their construction is often automatic, granting for their expansion with a variety of techniques. however, these techniques [3, 6, 16, 41, 45, 46, 61, 81, 87, 95, 97, 103, 105, 107] come with their own set of limitations, most notably how they provide close to no input about why they provided the results they did. In addition, the tools available to perform these tasks are limited due to the nature of their conception, created to test a specific technique and made available for replication purposes only by expert users, as opposed to being designed with usability and customization in mind.

According to the previous argumentation, we conclude that our hypothesis is the following:

> *Knowledge Graphs provide a way to structure information such that it provides semantic value to applications that rely on them and are widely used. They are inherently incomplete and can be expanded, however, the methods that do so are limited in two major ways, they lack explainability for the generated knowledge and are only usable for expert users, if at all.*

### 1.2.2   Thesis

A multitude of approaches have tackled the problem of KG completion [3, 6, 8, 46, 61, 82, 87, 97, 103] and reasoning [13, 14, 16, 41, 45, 81, 95, 105, 107]. Triple **classification** KG completion methods aim to classify new triples as true or false to include them in the graph, hence completing them, however, these methods lack explainability for their results and do not provide any knowledge apart from a certain degree of confidence. On the other hand KG **reasoning** approaches do provide an explanation for how they reached their results and they do so by applying a plethora of algorithms and techniques that are not without their own limitations. Their reward structures are based on the retropropagation of simple terminal rewards that require the learning Agent to reach a particular state to progress. They also suffer from highly variable inference

accuracy caused by the RL algorithms used. They often disregard the semantic value of the vectorial representations they use for the nodes and relations and only apply them as the input of NN classifiers. Finally, existing techniques could be updated or expanded by domain experts if their construction granted for it, however, the tools presented are almost always tailored for the only purpose of producing results for publication and nothing more, and this generally means making them hard to work with and expand.

In light of the previous reasoning, we conclude that our thesis is as follows:

*Knowledge graph reasoning can benefit from the development of novel reward structures that leverage the semantic value contained within the embedding representations of graph nodes and relations, more robust Reinforcement Learning algorithms that circumvent the high variance of the methodology of policy gradient methods and from a standardized approach to applying these techniques that focus on ease of expansion, modification and application to make it possible to implement new techniques in the future supported by the same set of tools while also providing support to any user.*

## 1.3    Summary of contributions

To prove our thesis, we have devised ***SpaceRL, our KG reasoning framework*** supported by Reinforcement Learning techniques.

SpaceRL is an end-to-end framework focused on customization and adaptability, it works with any dataset provided in the expected format. SpaceRL allows any user with any level of expertise to generate an intelligent agent able to reason over the KG of their choice in order to generate new knowledge in the form of a triple accompanied by a structured reasoned path.

The framework allows the users to customize it by permitting the tuning of hyperparameters, relying on previous experience with LSTM layers, generating a particular set of embeddings for the dataset being used, using multiple RL algorithms, reward structures and depth of reasoned paths.

It also allows for the deployment of a relation API in order to be a service provider for tuned models or to provide infrastructure for model generation to a third party, as well as offering a GUI in order to give inexperienced users a comprehensive view of the system for them to generate models and tune them as needed, without expert knowledge of the field. The system also offers visualization tools that give the user an overview of how the trained models choose the paths while in inference mode in order to better understand the decisions made.

SpaceRL is meant to be easily expanded, however, it also comes with multiple

options out of the box, we have mentioned in the introduction how current KG reasoning approaches are lacking in inference stability, which is why SpaceRL defaults to the usage of the Proximal Policy Optimization (PPO) reinforcement learning algorithm and comes out of the box with novel reward structures based on the properties of the embeddings for the evaluated KG. these options were used in our paper "SpaceRL-KG: Searching Paths Automatically Combining Embedding-based Rewards with Reinforcement Learning in Knowledge Graphs", currently under review in Expert systems with applications. The framework, complete with API and GUI has been sent to SoftwareX, and is currently awaiting to be reviewed by the editor.

## 1.4   Structure of this dissertation

This dissertation is structured as follows:

**Part I: Preface.**  It holds the introductory chapters to this dissertation, the one holding these words plus Chapter 2, where we go into detail about the justification that granted the works presented in this book, presenting the limitations of the current state-of-the-art approaches.


**Part II: Background Information.**  This Part gives insight into the topics necessary to better understand the proposals presented, In chapter 3 we give an overview of Knowledge graphs, how they are structured, constructed, applied and the challenges in the field.   In chapter 4, we introduce the representational techniques for knowledge graphs known as embeddings, how they work and several proposals that apply the different techniques used to generate them. In chapter 5 we give an introduction to Reinforcement Learning, how it can solve KG reasoning tasks and how it operates within the scope of a knowledge graph, the evolution of the techniques and algorithms and how each of them operates and some applications RL has had success on.

**Part III: Our Proposal.**  This Part reports on the contributions made in this dissertation. In chapter 6 we describe the methodology followed to create reinforcement learning agents and how they fared on several state-of-the-art datasets to generate reasoned paths following our new reward structure and use of modern RL algorithms. In chapter 7 we present our framework for RL-based KG reasoning tasks, how it works and the advantages it poses over creating new one-time use code for a proposal, how it can help expert users provide a service to others and how can be used by non-expert users.

**Part IV: Final Remarks.**  It contains Chapter 8, which concludes this dissertation and presents some possible future research directions.

# Chapter 2

# Motivation

*"The worthwhile problems are the ones you can really solve or help solve, the ones you can really contribute something to. No problem is too small or too trivial if we can really do something about it."*

— Richard Feynman

Knowledge Graph reasoning using reinforcement learning is an active topic of research, several proposals have emerged in recent years, however, they present drawbacks that hinder them. This chapter studies these proposals the problems each of them presents, and how that motivated our work. This chapter is organized as follows: Section 2.1 introduces it and provides the necessary background knowledge, Section 2.2 presents the problems that show up in KG reasoning, Section 2.3 analyzes the current approaches and their drawbacks, Section 2.4 explains how none of the existing proposals solves all practical problems at a time, Section 2.5 introduces our contributions and compares them with the existing proposals in the literature; finally, Section 2.6 summarizes the chapter.

## 2.1  Introduction

In recent years, there has been a growing interest of major tech companies in Knowledge Graphs (KGs) fueled by the proven efficacy of KGs in structuring and organizing information. Giants like Google, Facebook, and Microsoft adopted KGs as part of the core of their organizations, recognizing them as instrumental in organizing vast amounts of information, a new era of knowledge representation unfolds. This chapter explores the motivations driving the creation of this book, tracing the trajectory of the growing significance of KGs in information management.

Knowledge Graphs, with their inherent ability to capture complex relationships between entities, have proven to be indispensable tools for organizing information at an unprecedented scale. The capacity to structure data in a way that reflects the intricacies of real-world connections, offering a more nuanced understanding of the underlying domain.

However, the way in which KGs are automatically constructed leads to their inherent incompleteness, hence the main challenge for KGs lies in their augmentation with unexplicited information that can be inferred from itself, the process of Knowledge Graph Completion (KGC), in this way ensuring they are as comprehensive as possible.

KGC provides new knowledge to be incorporated into the KG, it provides no reason for it more than a numerical value, KG reasoning stands as a pivotal advancement in this matter. Unlike conventional completion approaches that primarily address missing links through direct imputation, KG reasoning introduces a more sophisticated layer of inference.

Through reasoning, the completion process transcends mere data imputation, dynamically synthesizing implicit connections within the graph, offering a more nuanced and comprehensive representation of the underlying knowledge, by leveraging Reinforcement Learning it is possible to generate intelligent agents capable of constructing paths of reasoned knowledge that offer insight into the facts being inserted into the graph.

In the literature, there are different proposals that address the problem of applying KG reasoning through RL for Knowledge graphs [14, 16, 41, 45, 95, 105, 107]. These proposals, however, are limited due to the application of the methods they propose. For this reason, furthering the strategies and methodologies for KG reasoning is our main purpose in this dissertation.

## 2.2  Problems

Knowledge graph completion is a complex task, more so if we focus on explaining the knowledge being generated. Obtaining said knowledge through reinforcement learning

algorithms presents its own set of challenges to add to that, and to be successful in practice, these challenges must be overcome. In this section, we present some of the problems that appear in the proposals focusing on this topic:

**(P1) Usage of embedding representations while not providing them:** Multiple proposals that tackle the problem of Knowledge Graph reasoning require being provided with the embedded representation of entities and relations on the graph, this forces users to generate these representations which can be a deterrent to the accessibility of the tools created. Embedded representations work by capturing meaningful semantic similarities among diverse elements of the graph and compacting them into a numerical N-dimensional vector; however, they are accompanied by a significant drawback, they are highly sensitive to alterations on the KG and must be re-generated if any change occurs to said KG. Using these embedded representations is a double-edged sword, it allows for a simple way to vectorize graph elements, but, it makes the proposals using them dependent on them and in general, they do not offer a way to generate them on the fly, adding another layer of complexity to the process.

**(P2) Hardcoded implementations of specific approaches or no implementation at all:** Existing KGC and KGR approaches focus only on generating data for publication purposes, this entails that the tools produced are not practical for usage outside of that scope, if they are even available, which is not always the case, they generally lack customizability, they require the embedding representation to be provided and they are cumbersome to expand or update. This reduces accessibility to these tools to users who seek further development in the field, users who want to make use of the tools but lack the required expertise to make them work, or users who want to expand them for their purposes.

**(P3) Knowledge graph completion offers low explainability of solutions provided:** One of the main problems with KGC which is intrinsic to the nature of the technique is the nonexistent explainability of the knowledge generated. KGC techniques generally fall under the umbrella of binary classifiers, they offer a simple yes or no answer to a query triple and it's then incorporated into the graph. Embedding techniques also offer a ranking of triples that fit a given query (h, r, ?) but also offer no reason behind them.

**(P4) Reward structured based on terminality:** Reasoning techniques attempt to teach a Reinforcement Learning agent to traverse graph nodes as states and relations as the possible actions. The training of this agent has been performed in a variation of a particular approach throughout the literature, the agent must

reach the target node and the rewards are retropropagated through the graph nodes visited. This requirement of reaching the end node before the policy network can begin to be modified causes high inference times

**(P5) KGR techniques use low convergence algorithms:** There exists a plethora of Reinforcement Learning algorithms, due to the nature of the reward structure (a binary reward upon reaching the destination node) it is a forced choice to use a retropropagation algorithm. These algorithms have a major drawback of having a low convergence rate for complex problems, in theory, if left ad-infinitum it would always converge on the correct solution, however, this is obviously impractical in a real-world scenario. Retropropagated algorithms tend to get stuck in local optimums as they have no awareness of the optimal path beforehand or have no metric to follow apart from the fact that they have successfully reached the state they were required to or not.

## 2.3  Analysis of current solutions

A number of proposals that work on Knowledge Graph Reasoning already exist in the literature. Table 2.1, displays the most relevant ones which will be discussed in the following parragraphs in further detail

Lao et al. [41] presented the Path Ranking Algorithm, it obtains reasoned paths connecting entities and ranks them based on a multitude of random walks performed over the knowledge graph and then tunes the weights in future random walks according to the result of previous inference.

Nickel et al. [61] devised RESCAL as a way to represent knowledge graphs by modeling the triples as tensors, where two modes of the tensor represented the entities and another mode the relation that connects them. By organizing the Graph in this way it can perform tensor factorization operations and predict the existence of triples by observing the rank-reduced reconstruction of the produced slice.

Bordes et al. [6] proposes a method to embed entities from a Knowledge Graph into an N-dimensional space, aligning semantical similarities with physical distance in this space. The approach involves learning embeddings that enable a meaningful arrangement of entities based on their semantic attributes. Notably, the method introduces a novel criterion for triple accuracy, evaluating it by analyzing the relative positions of entities in the embedded space, and linking semantic coherence with geometric proximity.

Wang et al. [103] improved upon this previous method by altering the way in which the position of the ranked relations was calculated, instead of linearly they performed a translation into a hyperplane based on the vector of the evaluated triple's

| Proposal | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| Bordes et al. [6] | ✓ | X | X | ? | ? |
| Cui et al. [14] | X | X | ✓ | X | ✓ |
| Das et al. [16] | ✓ | X | ✓ | ✓ | X |
| Lao et al. [41] | ? | X | ✓ | ? | ? |
| Lin et al. [45] | X | X | ✓ | X | X |
| Nickel et al. [61] | ? | X | X | ? | ? |
| Tiwari et al. [95] | ✓ | X | ✓ | ✓ | X |
| Vashishth et al. [100] | ? | X | X | ✓ | X |
| Wang et al. [103] | ✓ | X | X | ? | ? |
| Xian et al. [105] | X | X | ✓ | X | X |
| Xiong et al. [107] | X | X | ✓ | ✓ | X |

P1 = Reliance on Embedding representations and not providing them; P2 = Hardcoded implementations of specific approaches; P3 = No explainability of the knowledge generated (binary classifier/ranking); P4 = Reward structure based on terminality (if aplicable); P5 = Low convergence algorithms (if applicable)
✓ means the proposal does not have this problem, X means that it does suffer from it and ? means its not applicable for that proposal.

**Table 2.1:** Comparison of current proposals for KG reasoning and completion.

relation in the N-dimensional space, this paved the way for many other models which improved upon these ideas.

Xiong et al. [107] presented DeepPath, where pre-computed paths are evaluated by several metrics by a reinforcement learning algorithm, and ranked based on these metrics.

Das et al. [16] proposed MINERVA, a proposal on KG reasoning based on reinforcement learning, this proposal opened a path for others to follow, it was the first to propose the usage of graphs nodes as states, and relations as possible actions, they implemented it based on a simple terminal reward that retropropagated to the previous states following the REINFORCE implementation.

Lin et al. [45] implemented a multi-hop KGR proposal, this technique was the first to take on improving the reward structure presented by MINERVA, it reinforced the reward by comparing it to a pre-trained one-hop model which helped estimate the reward of evaluated facts and introduced the concept of action masking to force the agent to take paths that might not normally be explored.

Xian et al. [105] proposed an implementation that focuses on Recommendation systems via a method called PGPR which focuses on providing interpreted paths, relying also on a terminal reward complemented by a secondary NN that helps the inference during training in a dual manner.

Vashishth et al. [100] devised InteractE a proposal that also relies on dimensionalizing graph elements, then applying convolutional neural networks to them in order to perform operations on the features of these embedded representations instead of a translation operation for them to interact more (hence the name) causing a more rich solution for the destination node.

Tiwari et al. [95] presented a distance-aware reward system that addresses variable rewards at different graph positions by integrating graph self-attention mechanisms to capture entity information and combine it with a (GRU) for path retention.

Cui et al. [14] constructed an approach that focuses on anticipating the next step of the agent being trained with an "anticipation network", It treats the queries as a question and searches for an answer, using reward shaping and SAC.

## 2.4   Challenges

The proposals presented in the previous sections all have one or multiple drawbacks.

Regarding the usage of embeddings (P1), most if not all proposals based on reinforcement learning rely on these representations to operate, making them suffer from the problems that accompany them, moreover, they do not provide the user with a simple way to generate these embedded models further hindering their usability. Proposals that try to perform KG completion by leveraging translational models or variations of them also require these embeddings to be generated as it is the core of these proposals, this makes their results less interpretable.

Most of, if not all, proposals regarding the topics of KGC and KGR ignore the usability and expandability of their work (P2), they focus on presenting their implementations in a simple manner so that their results are replicable in a few command line prompts, however, this is not useful for researchers looking to further a topic or users trying to apply these works to some applications.

KGC approaches [6, 61, 100, 103] all are hindered by the lack of reason behind their approaches (P3), they compute either a ranking or classify a list of given triples

but lack the capacity to offer a sense of the results they provide, for this reason, results given are unverifiable by humans that are not experts on the domain being treated and will have to be trusted blindly.

But explainability is not everything, if approaches that do offer reason to the answers provided are bound to the same lacking methodology (P5) when obtaining it, they can become hard to train and get stuck in local optimums if the precise expected configurations are not attained by the users who try and make use of them, several proposals [16, 45, 95, 100, 105, 107], suffer from this and would benefit from leveraging more modern algorithms that prevent these local optimums from forming in the Policy NN.

In tandem with this problem is also the fact that the majority of proposals focus on altering how the policy learns and reacts by tweaking the same base terminal reward (P4). These policies provoke spurious paths to occur and several [14, 45, 105] try to mitigate with a plethora of solutions, masking actions, following pre-trained one-hop models to compare to for each of the actions, when a generalistic guided reward can avoid most of these problems while also leveraging terminality.

## 2.5   Our proposal

In order to mitigate the aforementioned problems, we present SpaceRL, our Knowledge Graph Reasoning framework, and end-to-end tool which provided a dataset outputs trained models able to perform KG reasoning on said KG (P3).

SpaceRL implements a novel, fundamentally different set of reward functions that exploit node embeddings, as well as the structural distance to the answer node, we reward actions that lead to nodes that are semantically similar to the target or are closer to it, without having to reach the answer node (P4). We also evaluate for the first time the application of the Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC) (P5)

SpaceRL combines the benefits from RL pathfinding with the power of representational embeddings to infer fairly long and explainable paths, useful for KG-based applications, and it can do so with on-the-fly embedding generation, which means that the KG embeddings are not a required input to the system (P1).

Our tool is highly configurable, allowing for reward calculation to be modified with a combination of several options while allowing the user to apply state-of-the-art RL algorithms out of the box (P2), namely Proximal Policy Optimization (PPO)[73] combined with Soft Actor-Critic (SAC)[29], which improve performance and help avoid reward plateaus while training.

Finally, SpaceRL aims to provide a versatile tool intended for users with different

levels of expertise, from novices to experts. It allows comprehensive and flexible customization for advanced users, who may prefer to install SpaceRL as a server for their local usage or to become a service provider for third parties.

SpaceRL offers RL model generation and usage as a service capability, either locally through its GUI or as a deployable REST API for third-party consumption.

## 2.6   Summary

In this chapter, we have elaborated on the motivation for this dissertation. We analyzed the problems present in Knowledge Graphs Reasoning and Completion and the current proposals that exist in the literature and concluded that none of these proposals solve all of these problems.

# Part II

# Background Information

# Chapter 3

# Knowledge Graphs

*"To know that we know what we know, and to know that we do not know what we do not know, that is true knowledge."*

— *Nicolaus Copernicus*

Knowledge Graphs (KGs) are a way to structure information by representing them into a series of interconnected facts and relations between them. This chapter gives some context around them. It is divided into the following sections: Section 3.1 presents a brief history of Knowledge Graphs and their adoption. Section 3.2 goes over some of the more relevant KGs currently. Section 3.3 goes about some of the many applications KGs are currently being used for. Section 3.4 reflects the current open challenges for KGs. Finally, Section 3.5 summarizes and concludes this chapter.

# 3.1   Introduction

Before the emergence of knowledge graphs, information was predominantly stored in traditional databases and represented in tabular formats. While these databases were effective at storing structured data, they lacked the capacity to capture the complex relationships and semantics inherent in real-world information.

The concept of knowledge graphs can be traced back to the early days of Artificial Intelligence (AI) research. In the 1960s and 1970s, researchers began exploring methods to represent knowledge in a form that could be understood and utilized by computers. Early endeavors focused on semantic networks, which used nodes to represent concepts and edges to denote relationships between them.

The advent of the World Wide Web in the 1990s brought about an explosion of digital information. As the volume of web content grew, so did the need for more sophisticated methods of organizing and extracting knowledge from this vast repository. This led to the development of the Semantic Web, a vision championed by Sir Tim Berners-Lee, which aimed to make web content machine-understandable.

A pivotal milestone in the evolution of knowledge graphs was the introduction of the Resource Description Framework (RDF) in the late 1990s. RDF provided a standardized way to describe resources on the web and establish links between them. This laid the foundation for the creation of linked data, which allows for the interconnection of disparate datasets on the web.

In 2012, Google introduced the term "Knowledge Graph" as a central element of their search engine. Google's Knowledge Graph aimed to enhance search results by providing contextual information about entities and their relationships. This marked a significant shift towards a more semantically enriched approach to information retrieval.

Rather than a simple collection of relations between names of entities, they started to be seen as a rich, interconnected structure of elements (*"things, not strings"*) with an enormous potential for practical and commercial applications. Many other large companies the likes of Amazon, Facebook, Microsoft and eBay soon followed suit and the term Knowledge Graph (KG) rose to the popularity it still enjoys nowadays, replacing the denomination "Knowledge Base".

Today, knowledge graphs have become a cornerstone of various AI applications, including natural language processing, recommendation systems, and data integration. Their ability to represent complex relationships and semantic context has made them an invaluable tool in the era of big data and advanced machine learning techniques.

## 3.2   Modern Knowledge Graphs

- **DBpedia** is a community-driven knowledge graph that extracts structured information from Wikipedia articles. It covers a wide range of topics and provides structured data about entities, their attributes, and relationships. DBpedia utilizes natural language processing and information extraction techniques to parse Wikipedia articles and convert unstructured text into a structured RDF format.



**Figure 3.1:** The entity *Keanu Reeves* in DBpedia

- **YAGO** (Yet Another Great Ontology) is a knowledge graph that combines data from Wikipedia, WordNet, and GeoNames. It provides a comprehensive representation of entities and their relationships. Created using automated extraction techniques and ontological alignment to integrate information from multiple sources.
- **Wikidata** is a collaborative, multilingual knowledge graph maintained by the Wikimedia Foundation. It serves as a central repository of structured data for Wikimedia projects and beyond, containing information about a diverse set of topics it is built by a global community of volunteers who contribute, edit, and curate data using a web-based user interface.

Statements

| instance of | human |
|---|---|
| | ▸ 2 references |

| image |  |
|---|---|
| | Reunião com o ator norte-americano Keanu Reeves (46806576944) (cropped).jpg |
| | 1,329 × 1,790; 532 KB |
| | ▾ 0 references |

| sex or gender | male |
|---|---|
| | ▸ 5 references |

| country of citizenship | Canada |
|---|---|
| | ▾ 0 references |

**Figure 3.2:** The entity *Keanu Reeves* in Wikidata

- **UMLS (Unified Medical Language System)** is a comprehensive ontology and knowledge graph for the biomedical domain. It integrates terminology and data from various biomedical sources. UMLS is built through a combination of manual curation by domain experts and automated processes for integrating and mapping different terminologies.
- **Freebase (Now Part of Wikidata)** was a large-scale knowledge graph acquired by Google in 2010 and later integrated into Wikidata. It contained structured data on a wide array of topics, including people, places, and concepts. It was constructed through a combination of automated data extraction, crowd-sourcing, and expert curation.
- **WordNet** is a lexical database of the English language, organized in a semantic network. It groups English words into sets of synonyms (synsets) and provides short, general definitions. WordNet is manually constructed by lexicographers

who define word meanings and establish semantic relationships between words.

- **NELL (Never-Ending Language Learning)** is a project aimed at creating a machine learning system that continuously learns to extract structured information from the web. It aims to discover new facts about entities over time it uses a combination of natural language processing, machine learning, and information extraction techniques to learn and update its knowledge base.

## 3.3   Applications

Knowledge graphs, as we explored in the preceding chapter, are structured representations of information that emphasize the interconnectedness of entities, attributes, and relationships. This section delves into the practical applications and diverse utility of knowledge graphs across a spectrum of domains. The power of knowledge graphs lies not only in their capacity to organize information but also in their ability to unlock a deeper layer of context, semantics, and relationships within data.

They have emerged as a foundational tool in modern information processing and artificial intelligence. In this chapter, we delve into the diverse and powerful applications of knowledge graphs across various domains. From enhancing search engines to revolutionizing recommendation systems, knowledge graphs play a pivotal role in extracting valuable insights and enabling intelligent decision-making.

This chapter explores five prominent applications of knowledge graphs, each showcasing their versatility and impact in different realms of information processing, such as information retrieval, data analysis, and decision support. We illustrate how knowledge graphs have transformed traditional approaches, enabling more accurate, contextually aware, and personalized interactions with data.

- **Semantic Search and Information Retrieval**: Knowledge Graphs provide a structured foundation for deducing the context between entities by understanding the intricate relationships between them. Unlike traditional keyword-based searches, semantic search engines, empowered by knowledge graphs, consider the deeper meaning and connections within the data, they ensure that search results are more accurate and contextually meaningful. This is especially crucial in fields like healthcare, legal research, and scientific studies where precision in information retrieval is essential. Through semantic search, users can navigate complex datasets with greater precision, uncovering insights that might be missed in traditional keyword-based searches.

  For instance, in a healthcare knowledge graph, the relationships between symptoms, diagnoses, and treatments allow for more precise and contextually relevant search results. This is vital in scenarios where accuracy and depth of information retrieval are paramount.

- **Recommendation Systems**: Knowledge graphs play a pivotal role in recommendation systems, revolutionizing how personalized suggestions are generated. Recommendation systems, at their core, rely on understanding user behavior and preferences. Knowledge graphs provide the structured foundation needed to model these relationships effectively. By incorporating nuanced connections, recommendation systems can go beyond surface-level patterns, uncovering unexpected associations that lead to more meaningful and relevant suggestions. By modeling rich semantic connections between users, items, and their attributes, knowledge graphs enhance the accuracy and relevance of recommendations.

  This is particularly crucial in domains like content streaming platforms, e-commerce, and content curation, where tailoring suggestions to individual tastes can significantly enhance the user experience. Knowledge Graphs elevate the accuracy and effectiveness of recommendation engines. This results in more engaged users, higher conversion rates, and ultimately, a more satisfying user experience.

- **Natural Language Processing (NLP)**: Knowledge graphs provide a structured framework for representing and understanding textual information which significantly enhances natural language processing (NLP) tasks. It allows for more accurate and nuanced language understanding, leading to improved tasks such as entity recognition, relationship extraction, and question-answering.

  By incorporating semantic relationships between entities and their attributes, knowledge graphs enrich language understanding. They serve as a fundamental tool for NLP applications. They allow for a more nuanced analysis of textual data by considering not just individual words, but also their relationships within a broader context. Knowledge graphs enable NLP systems to go beyond basic language processing, enabling them to grasp the complexities of human communication more effectively.

- **Contextual Analysis and Decision Support**: Contextual analysis involves examining information within the broader framework of its surroundings, considering the environment, relationships, and relevant circumstances that influence its meaning or interpretation. It is crucial to understand the full significance of data. It helps in avoiding misinterpretations that can occur when information is considered in isolation. By taking into account the surrounding context, analysts can gain deeper insights and make more informed decisions.

  Knowledge graphs enhance contextual analysis by providing a structured framework for representing relationships and contextual information between entities. These graphs incorporate temporal, hierarchical, and semantic dimensions, allowing for a comprehensive understanding of data within its broader context. Moreover, knowledge graphs facilitate context-aware queries, enabling users to retrieve information based on specific contextual constraints.

This not only supports decision-making but also empowers analysts to uncover hidden patterns and dependencies within the data, ultimately leading to more informed and effective analyses.

- **Healthcare**: In healthcare, knowledge graphs serve as a powerful tool for representing, organizing, and analyzing complex medical information. They provide a structured framework that captures relationships between medical entities, their attributes, and contextual information. For instance, a healthcare knowledge graph can model the relationships between symptoms, diagnoses, treatments, medications, and patient histories. This enables comprehensive and contextually rich representations of medical data, which is crucial for accurate diagnosis, treatment planning, and research.

  One of the key strengths of knowledge graphs in healthcare lies in their ability to integrate diverse sources of medical information. They can incorporate data from electronic health records (EHRs), medical literature, clinical trials, and other healthcare databases. This integration enables a holistic view of a patient's medical history and facilitates evidence-based decision-making by healthcare professionals.

  Furthermore, knowledge graphs support clinical decision support systems (CDSS) by providing a structured foundation for generating patient-specific recommendations and alerts. For example, a CDSS built on a healthcare knowledge graph can analyze a patient's symptoms, medical history, and medication interactions to offer tailored treatment suggestions. This not only enhances the quality of care but also helps healthcare providers stay up-to-date with the latest medical knowledge.

  In addition, knowledge graphs are invaluable for medical research and innovation. They enable researchers to explore relationships between genetic factors, diseases, treatments, and outcomes. This can lead to discoveries about disease mechanisms, personalized medicine approaches, and the development of new therapies.

## 3.4 Open challenges

While knowledge graphs offer a powerful framework for organizing and leveraging structured data, they are not without their complexities and hurdles. In this section, we turn our attention to the open challenges that persist in the field of knowledge graphs. These challenges encompass three critical domains: integration, completion, and reasoning. Addressing these issues is essential for unlocking the full potential of knowledge graphs in diverse applications. Let's delve into each of these challenges and explore the research frontiers that seek to overcome them.

### 3.4.1 Integration - Joining diverse data sources

Knowledge graph integration addresses the crucial task of amalgamating information from disparate sources into a unified, coherent representation. This process is instrumental in creating comprehensive knowledge graphs that encapsulate a wide array of domains. Ontology-based integration employs predefined schemas to structure data, ensuring compatibility across diverse sources. Federated approaches, on the other hand, enable querying across distributed databases, extracting relevant information from each source. Schema matching and alignment techniques aim to identify correspondences between different schemas, facilitating the mapping of data elements.

These approaches collectively form the bedrock of knowledge graph integration, enabling the creation of holistic, interconnected representations. Integration is particularly challenging when dealing with multilingual data, as language nuances add an additional layer of complexity. The global nature of information demands knowledge graphs that transcend language barriers. Multilingual knowledge graphs integrate data from various linguistic sources, enabling a truly global perspective. Machine translation techniques play a pivotal role, in facilitating the transformation of content between languages. Additionally, cross-lingual entity linking ensures that entities mentioned in different languages are correctly aligned, enhancing the coherence and richness of the knowledge graph.

Ambiguity in entity references poses a significant challenge in knowledge graph integration. Entity disambiguation techniques strive to resolve this ambiguity by identifying and linking mentions of the same entity across different data sources. Methods based on contextual information, entity co-occurrence, and semantic similarity have shown promise in disambiguating entities accurately.

### 3.4.2 Completion - Finding missing information

Knowledge graphs are prone to incompleteness due to the inherent challenges in capturing all facets of real-world information. Incomplete knowledge graphs often arise from limitations in data collection, varying levels of detail in different domains, or evolving sources of information.

Additionally, semantic gaps may emerge when attempting to represent complex relationships or when dealing with ambiguous entities. Knowledge graph completion plays a vital role in mitigating these shortcomings in order to refine the graph's structure to empower applications, by providing a more comprehensive and accurate representation of the underlying domain.

Knowledge graph completion is the task of predicting missing or incomplete information within a knowledge graph it is an indispensable task in enhancing their

comprehensiveness and usability. The process of completion involves inferring new relationships or attributes for existing entities, ultimately enriching the graph's representation. By predicting these missing links, knowledge graph completion enables more accurate querying, reasoning, and recommendation within the graph.

Knowledge graph completion is executed through a variety of techniques, each designed to infer missing relationships or attributes within the graph. These approaches leverage the existing structure of the knowledge graph, exploiting patterns and dependencies to make accurate predictions. One common strategy involves embedding-based models, which represent entities and relationships as vectors in a continuous space. These models aim to learn embeddings that capture the underlying semantics of the graph, allowing for the extrapolation of missing information. Additionally, rule-based methods utilize logical rules to deduce new relationships based on existing ones. These rules can be hand-crafted or learned from the data, providing a structured approach to completion.

Another prevalent approach in knowledge graph completion involves utilizing graph neural networks (GNNs). These specialized neural networks operate directly on the graph structure, enabling the propagation of information through nodes and edges. By leveraging the local and global connectivity patterns, GNNs can uncover complex dependencies and infer missing links effectively. Additionally, path-based techniques explore sequences of relationships within the graph to identify latent connections between entities. These methods analyze the paths connecting entities and use them to make predictions about missing links.

Once the new information has been inferred it can be used to enrich the KG by adding the missing triples obtained from this process back into the KG. This completion task can be performed every time the KG is extended due to temporal shifts in the information it holds, this way the KG will always be up to date.

### 3.4.3 Reasoning - Extracting deeper insights

In regard to knowledge reasoning, different nomenclatures have been incepted in academic circles. Initially, reasoning is the process of analyzing, synthesizing and deciding on several matters. It begins with collecting data in the form of facts and discovering interrelationships between them to then develop new insights. In short, reasoning is the process of "drawing conclusions from existing facts by the rules"[88].

The development of reasoning techniques also affected their definition, now including the capacity to understand things, apply logic, and validate elements based on existing knowledge. In other words, the mechanism behind inferring new knowledge is based on the existing facts and logical rules. In general, knowledge reasoning is ultimately defined as the process of using *known* knowledge to infer *new* knowledge.

The internet resulted in dataset sizes exploding, making it hard if not impossible to apply traditional methods. For this reason, data-driven machine reasoning methods have gradually become the most popular approach when it comes to knowledge reasoning research.

With the development of knowledge graphs, reasoning over them has gained traction in research. Its goal is to use machine learning methods to infer potential relations between entity pairs and identify erroneous knowledge automatically with the purpose of complementing KGs according to extracted logical rules that justify the inferences.

Several KG reasoning techniques exist, we focus on the most promising, Inference and Embedding-based techniques. Inference techniques can perform domain knowledge reasoning by modeling domain rules and specific knowledge which can support automatic decision-making, data mining and link prediction. They apply logical rules and algorithms to traverse, analyze, and connect entities and relationships within the knowledge graph. These rules encode domain-specific knowledge and can be hand-crafted or learned from data.

> Ejemplo de Inference techniques, as dani puts it: Estaría bien un ejemplo tipo 'por ejemplo, una técnica podría observar que la ocurrencia de la relación "abuelo" está, desde el punto de vista estadístico, altamente correlacionada con la presencia de un camino entre las entidades con dos aristas de la relación "padre de" y usar esta regla para insertar una arista de "abuelo" cuando se observa ese camino'

Embedding-based reasoning leverages continuous vector representations of entities and relationships to perform operations that approximate logical reasoning. Graph neural networks (GNNs) [83] are a powerful approach that directly operates on the graph structure, enabling the propagation of information and the capture of complex dependencies.

Knowledge graph reasoning offers significant improvements over simple completion tasks. While completion focuses on predicting missing links or attributes, reasoning enables a deeper level of understanding by inferring complex relationships and uncovering latent patterns

> again focus on a running example here, dani puts its as : Creo de nuevo que aquí viene bien un ejemplo. Completion hace referencia a simplemente predecir, por ejemplo, un enlace, sin dar justificación. Por ejemplo, <bla bla bla>. Mientras que el término reasoning implica que esto se hace proporcionando el razonamiento que se ha seguido para llegar a esa conclusión. Por ejemplo, por la presencia de ciertas relaciones o caminos entre las entidades involucradas

. It goes beyond the immediate graph structure to extract higher-order knowledge, making it invaluable in tasks that require sophisticated analysis and decision-making.

Our approach, SpaceRL, places a particular emphasis on knowledge graph reasoning. By incorporating reinforcement learning techniques, we aim to enhance the reasoning capabilities of agents navigating knowledge graphs. This allows for dynamic exploration and exploitation of the graph's structure, leading to more effective inferences and insights. In future explorations, we intend to delve deeper into this critical aspect, refining and extending SpaceRL to unlock even greater potential in knowledge graph reasoning.

## 3.5   Summary

This chapter serves as an introduction to Knowledge Graphs, and initiates the reader with a portrayal of their historical evolution and fundamental attributes. Additionally, it offers a comprehensive overview of noteworthy Knowledge Graphs currently prevalent. The discourse extends to an exploration of pragmatic applications emanating from Knowledge Graphs, elucidating their pervasive influence on our daily activities. Concluding this exposition, the chapter scrutinizes enduring challenges inherent in the domain of Knowledge Graphs, specifically focusing on unresolved issues such as integration, completion and reasoning.

# Chapter 4

# Knowledge Graph Embeddings

*"Computer Science is a science of abstraction -creating the right model for a problem and devising the appropriate mechanizable techniques to solve it"*

— Alfred Aho

R epresenting the information in knowledge graphs in a way that is usable for machine learning techniques became one of the main focuses of research after their inception. These representations originated from several KG completion techniques in which numerical vectors representing the entities and relations of the KG came as a byproduct of applying them. In this chapter, we showcase the improvements introduced by several of these techniques. It is structured as follows: Section 4.1 provides an introduction to the matter, Section 4.2 introduces the methods that leverage translation operations, Section 4.3 elaborates on the methods which focus on matching information semantically, Section 4.4 discusses the proposals which focus on the use of tensor representations of the KG; Section 4.5 overviews the methods which make use of neural network in their approaches, and finally, Section 4.6 provides a summary of the contents of the chapter.

# 4.1 Introduction

Embeddings are a form of representation of information that gathers the semantic, contextual, relational and modular data within an information node and converts it into a numerical representation, as a N-dimensional vector.

In particular KG embeddings represent the nodes and relations in the graph which can be captured either **semantically** by capturing the information held within the node or relation for example, a bag of words vector, **positionally** by studying the connections between graph nodes with one one of the multiple techniques described in the following sections, or with a combined approach, an enriched positional vector.

Knowledge graph embeddings aid in representing information within a KG in a way that is understandable for Machine learning tasks. These embeddings provide a compact, continuous vector representation for entities and relationships, allowing for efficient computational operations and meaningful interpretations.

These vectors capture the semantic nuances of the entities and the connections between them. In essence, knowledge graph embeddings serve as a bridge between the discrete, symbolic world of knowledge graphs and the continuous, numerical space of machine learning algorithms.

The process of obtaining knowledge graph embeddings involves mapping entities and relationships to low-dimensional vector spaces. This transformation is designed to preserve the essential structural and semantic information of the knowledge graph. Entities are represented as points in this vector space, while relationships are encoded as transformations that operate on entity vectors.

The goal is to position entities and relationships in such a way that their geometric relationships in the embedding space reflect the underlying semantic relationships in the knowledge graph.

For example, given the triple (h, r, t) by adding the vectorial representations of the connecting relation "r" to its origin node "h" the result should be the destination node "t" of that particular triple and be as furthest as possible from other nodes that are dissimilar to "t". The aim of these techniques is to maximize this similarity factor and to do so they iterate over the graph multiple times altering the values of these vectors untill they reach a satisfactory result.

In this manner, knowledge graph embeddings serve as a powerful tool for capturing the rich tapestry of connections, dependencies, and contextual information present in knowledge graphs and allowing this information to be passed to machine learning models which can solve a plethora of problems related to these powerful data linking tools.

Along the following sections the following nomenclature will be followed, where,

"h" denotes a head entity, "r" denotes the relation that connects them and "t" denotes a tail entity making (h, r, t) a triple in the graph. $\mathcal{E}$ is the set of all entities and $\mathcal{R}$ is the set of all relations in the graph.

## 4.2 Translation models

Translation models usually use distance-based functions to define the scoring function for link prediction tasks, the general idea behind these models is that if we apply a translation operation based on a relation "r" to an entity "h" represented by a low dimensional vector the result should be close to its destination entity "t" in the triple (h, r, t) which is considered correct as it is in the graph.

TransE [6] is a well-known, early and simple model that regards a relation as a translation from a head entity to a tail entity, used for link prediction it generates embeddings representations for entities and relations in KGs, it uses the relation representation to apply a translation from the head entity to a tail entity. It considers the uncertainties of entities and relations by using a probability function.
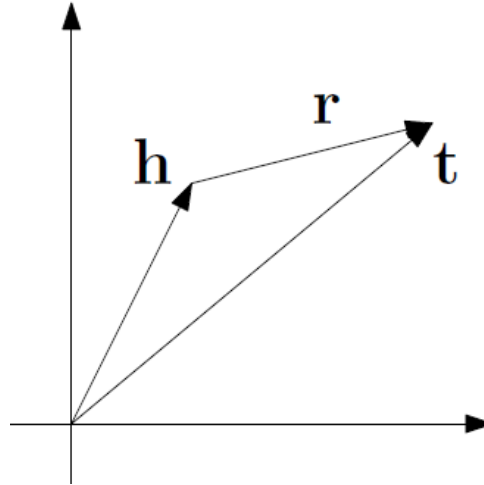


**Figure 4.1:** TransE [6] representation in 2D Space

The basic idea behind this model is that the connecting relation between the two entity representations makes it so $h + r \approx t$ as shown by the figure 4.1, It uses the distance scoring function defined by 4.1 accordingly.

$$f(h, t) = ||h + r - t||_1 \tag{4.1}$$

TransE is the earliest translation-based embedding model, and it has difficulty dealing with multi-relational graphs. It is limited by its simple translation operation as well as its lack of a discrimination policy for all kinds of relations.

Improving on TransE, TransH [103] introduces the concept of hyperplane translations. In order to overcome the problems of the former when it comes to

modeling reflexive multi origin or multidestination relations (one-to-many, many-to-one, many-to-many) this model enables an entity to have distributed representations when involved in different relations.
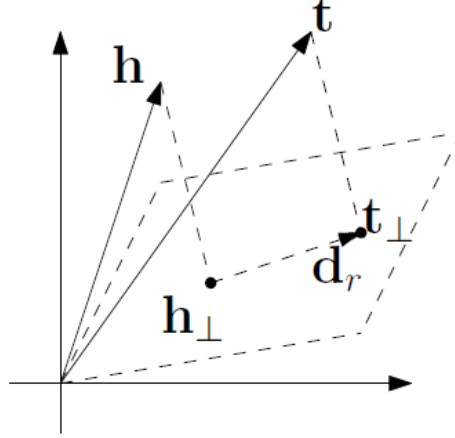


**Figure 4.2:** TransH [103] representation in 2D Space

As illustrated in Figure 4.2, the relation translation vector $d_r$ is contained within the plane rather than being connected directly such as in TransE.

for a triple (h, r, t), the embedding h and t are first projected to the plane ($h\perp$, $t\perp$). They are then connected by a translation vector $d_r$ in the hyperplane. The scoring function is described in Equation 4.2

$$f(h,t) = ||(h - w_r^\tau | h w_r) + d_r - (t - w_r^\tau t w_r)||_2^2 \qquad (4.2)$$

$$w_r, d_r \in \mathbb{R}$$

Improving on the TransH model appears TransR[46] which innovates by using a relation-specific space to handle different relations. As an entity may have multiple semantical interpretations different relations focus on different ones. TransR models entities and relations in distinct spaces, entity and relation spaces and performs translation in the corresponding relation space.

Figure 4.3 shows a simple rendition of the TransR model, it depicts how for a given triple (h, r, t), entities "h" and "t" are projected into the relation space for relation "r", then a translation is performed in said vectorial space. This relation-specific space also shows how other entities appear further away from the source triple as they are not close semantically for that particular relation space. TransR scoring function 4.3 is very closely related to TransE scoring function with the key difference that is performed in a different space.

$$f(h,t) = ||h_r + r - t_r||_2^2 \qquad (4.3)$$
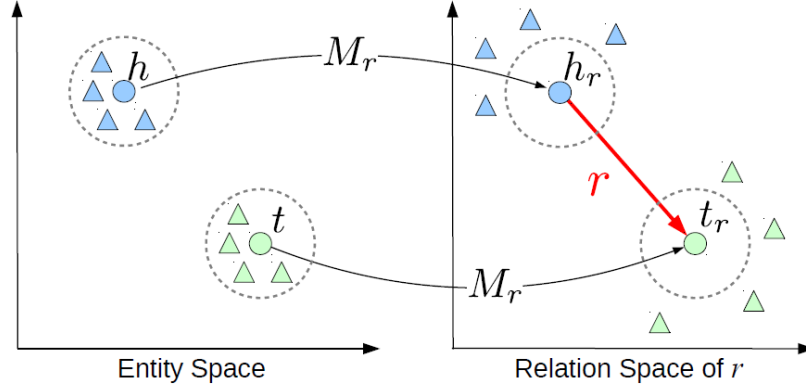
**Figure 4.3:** TransR[46] representation in 2D Space

Several iterations following the recipe of Trans "plus-a-leter" model have been proposed as improvements to the previous methods, all of them iterating on the idea of decoupling entity and relations interactions to then perform a translation operation of some sort.

To conclude this section RotatE[87] presents an improvement from previous techniques by defining each relation as a rotation from the "h" to "t" in a complex plane and the same vector space as shown in figure 4.4.
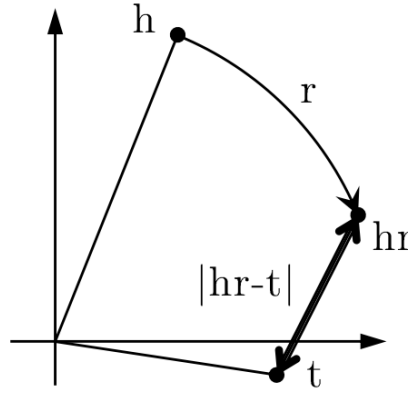


**Figure 4.4:** RotatE [87] in a 2D plane

RotatE maps the head and tail entities "h" and "t" to complex embeddings then, it defines a mapping function induced by each relation "r" as a rotation from "h" to "t" where $t = h \circ r, where |r_i| = 1$ and $\circ$ is the Hadmard product. By defining each relation as a rotation in the complex vector spaces, RotatE can model and infer relation patterns such as symmetry/antisymmetry, inversion and composition.

$$f(h,t) = ||h \circ r - t|| \tag{4.4}$$

The distance function of RotatE 4.4 corresponds to a counterclockwise rotation by

$\theta_r, i$ radians about the origin of the complex plane, and only affects the phases of the entity embeddings in the complex vector space.

## 4.3    Semantic information models

Semantic information-based models usually use similarity-based functions to define scoring functions for traditional semantic-matching models or introduce additional information to mine more knowledge for recently proposed models.

Traditional models match the latent semantics of entities and relation embeddings to measure the plausibility of a triple, however, these models suffer from high computational complexity.

More recent models fuse various additional information to obtain better performance to mine deeper semantic information at the bottoms of graphs. The additional information includes path information, order information, concepts, entity attributes, entity types and so on.

Word2vec[54] goal is learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary by using distributed representations of words learned by neural networks following stochastic gradient descent methods and backpropagation.
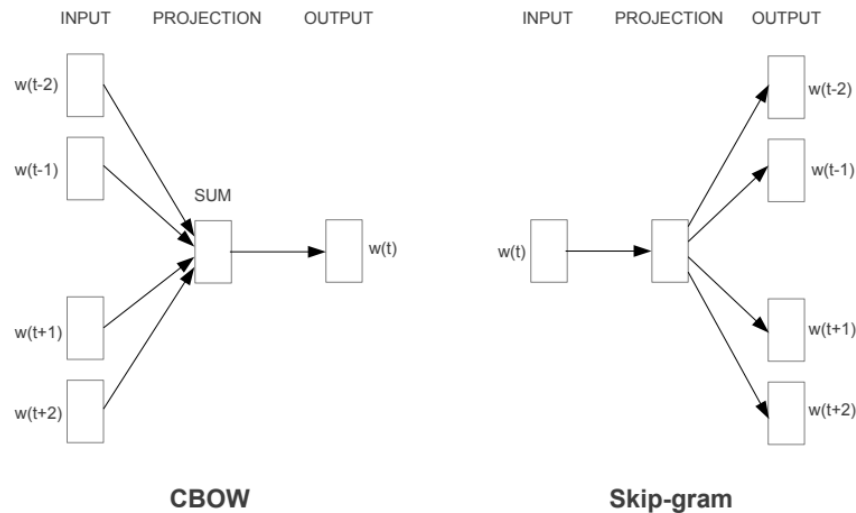


**Figure 4.5:** Word2Vec[54] models

It consists of two methods, a continuous bag-of-words model and a skip-gram model. The first of the two closely resembles a feedforward NNLM (Neural Network Language Model), where the projection intermediate layer is altered by all words in the input, therefore, all words alter the composition of the NN and their vectors are averaged.

The second model tries to maximize the classification of a word based on the sentence it is contained within. the entire context sentence is fed into a log-linear classifier which tries to predict words before and after the current word.

GlovE [65] appears as a model able to capture a global corpus of word statistics directly by using global matrix factorization and local context window methods. It is designed to capture semantic relationships between words by learning vector representations that encode the statistical information of word co-occurrence in a large corpus.

GloVe has become a popular choice for word embeddings in NLP due to its effectiveness in capturing semantic relationships and its relatively efficient training process.

GloVe starts by constructing a word co-occurrence matrix based on a given corpus. Each entry (i, j) in the matrix represents the number of times the word "i" co-occurs with the word "j" within a certain context window.

Then it trains in order to learn vector representations such that the dot product of these vectors corresponds to the logarithm of the observed word co-occurrence probabilities. The objective function of GloVe aims to minimize the difference between the dot product of word vectors and the logarithm of the co-occurrence probabilities.

The learned vectors represent words in a continuous vector space, where the distance and direction between vectors capture semantic relationships. Words with similar meanings or that often appear in similar contexts will have similar vector representations.

GloVe has been praised for its ability to capture meaningful semantic relationships between words. The resulting word embeddings often exhibit interesting properties, such as linear relationships that represent semantic analogies (e.g., "king" - "man" + "woman" $\simeq$ "queen").

## 4.4  Tensor Factorization models

Tensor factorization tries to solve KG completion tasks by relying on the fact that a Knowledge Graph can be represented as tensors where the relational data can be represented as a 0, 1-valued third-order tensor Y, and, if a relation (h, r, t) is true it meets $Y_{h,r,t} = 1$, KGC can be framed as a 3rd-order binary tensor completion problem.

The first approach to take advantage of the tensors being able to represent a Knowledge Graph was RESCAL [61], which takes the inherent structure of relational data into account by operating over this tensor representation.

RESCAL applies a decomposition into directional components approach which is capable of detecting correlations between multiple interconnected nodes. It decomposes
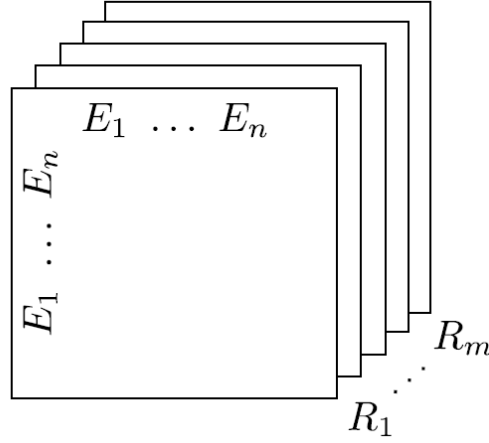
**Figure 4.6:** Tensor representation of a Knowledge Graph [76] with entities $E_1, ..., E_n$ and relation $R_1, ..., R_n$

them into a 3-way tensor as shown in figure 4.6, two dimensions are generated by the concatenated entity vectors and the third represents the relations matrix. In RESCAL, entities are expressed as vectors $v \in \mathbb{R}^d$ and relations are expressed as matrices $M \in \mathbb{R}^{d \times d}$ to calculate the score of a fact (h, r, t) a bilinear function is applied as defined by equation 4.5.

$$s(h, r, t) = v_h^T M_r v_t \tag{4.5}$$

where $vh, vt \in \mathbb{R}^d$ are entity embeddings, and $M_r \in \mathbb{R}^{d \times d}$ is the matrix associated with the relations between them.

Continuing the work on tensor factorization we find DistMult [110], an approach that tries to learn representations for entities and relations via a neural network in which the first layer projects a pair of input entities to low dimensional vectors, and the second layer combines them into an input of the bilinear scoring function 4.6 which represents the relation between them looking to maximize this score.

$$g_r^b(e1, e2) = e_1^T W_r e_2 \tag{4.6}$$

where $e_1$ and $e_2$ are the learned entity vectors and $W_r$ is the tensor operator.

Furthering this line of work is complEx [97]. As suggested by the name instead of using embeddings containing real numbers it furthers the idea of complex number embeddings, similarly to RotatE. ComplEx argues that the standard dot product between embeddings can be a very effective composition function when using complex vectors.

This instance of the dot product involves the conjugate-transpose of one of the two

vectors. As a consequence, the dot product is not symmetric anymore, and facts about antisymmetric relations can receive different scores depending on the ordering of the entities involved.

Thus complex vectors can effectively capture antisymmetric relations while retaining the efficiency benefits of the dot product, that is linearity in both space and time complexity.

A key difference with the RotatE model which also focuses on these complex spaces is that ComplEx applies a tensor factorization approach in a bilinear way, which means that the same entity will have two different embedding vectors, depending on whether it appears as the subject or the object of a relation.

Finally the TuckER [3] model named after the author of Tucker decomposition [98] applies the principles of said technique to the tensor factorization model, by decomposing a tensor into a core tensor multiplied by a matrix along each mode as shown in figure 4.7, TuckER generalizes the problem presented in previous approaches.
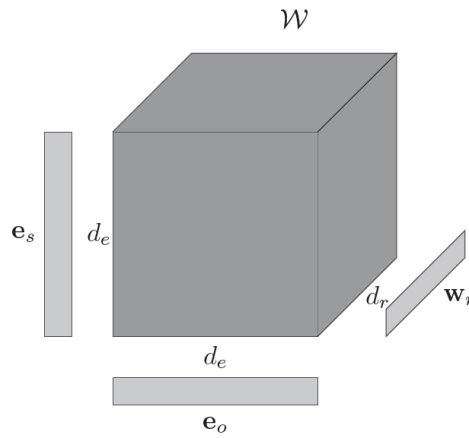


**Figure 4.7:** TuckER [3] architecture

Using the Tucker decomposition for link prediction on the tensor representation of a knowledge graph, with entity embedding matrix $\mathcal{E}$ relation embedding matrix $\mathcal{R}$ the scoring function for this model is as shown in figure 4.7

$$\theta(h, r, t) = \mathbb{W}x_1 h x_2 w_r x_3 t \tag{4.7}$$

where h, t are the rows of matrix $\mathcal{E}$ representing the subject and object entities vectors, $w_r$ the rows of the relation matrix and $\mathbb{W}$ is the core tensor to predict.

## 4.5   Neural network-based models

Neural networks can intelligently capture the semantic features of entities and relations and reasonably model the semantic relationships between discrete entities, which can help learn more accurate embeddings in the context of Knowledge Graphs problems.

The improvement of Deep Neural Networks (DNNs), Recursive Neural Networks (RNNs) or Graph Neural Networks(GNNs) along the previously discussed techniques made it an obvious path for research where NN models were tailored to generating these embedding representations of KG components. However, NN models apply non-linear transformations to the data they are provided in which they lose explainability in the generative process.

One of the initial approaches for this method is Neural Tensor Networks (NTN) [81] a link prediction method which replaces a standard linear neural network layer with a bilinear tensor layer that directly relates two entity vectors across multiple dimensions making it so entities are represented as an average of their constituting word vectors. The model computes a score of how likely it is that two entities are in a certain relationship by the following NTN-based function shown in figure 4.8



$$U^T f(\ e_1^T\ W^{[1:2]}\ e_2\ +\ V \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b\ )$$
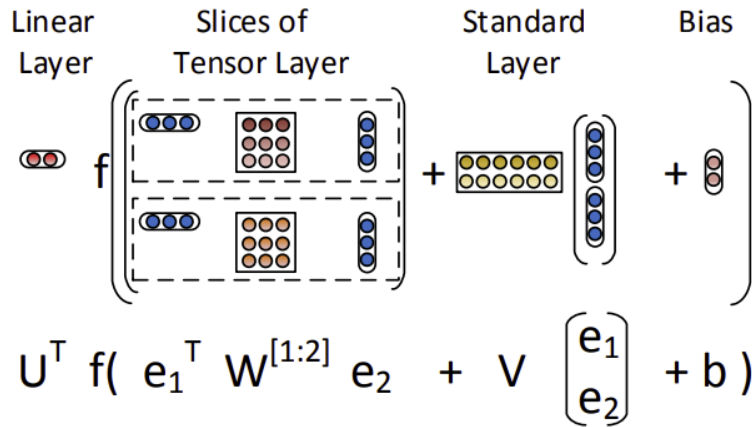
**Figure 4.8:** NTN layer [81]

ConvE [18] iterates upon the idea presented by NTN by using a multi-layer convolutional network model in order to be more parameter efficient than its predecessors and competitors such as DistMult.

ConvE utilizes 2D convolutions over embeddings to predict missing links in knowledge graphs. It is the simplest multi-layer convolutional architecture for link prediction and is defined by a single convolution layer, a projection layer to the embedding dimension, and an inner product layer.

The use of a 2-dimensional convolution layer increases the expressiveness of the model as it increases the number of interaction points, thus being able to extract more feature interactions between two entity embeddings.

InteractE [100] improves upon ConvE by providing three key ideas 1)feature permutation, 2) a novel "checkered" feature reshaping and 3)circular convolution, sinceas the expressiveness of a model can be enhanced by increasing the possible interactions between embeddings.
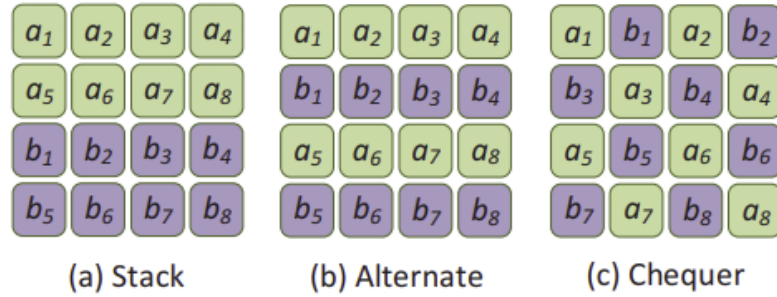


**Figure 4.9:** InteractE checkered pattern [100] for tensor reshaping.

*Feature Permutation*: Instead of a fixed order for inputs InteractE allows for a mutable input order, increasing the interaction between entity embeddings by allowing permutation between these inputs. However, the number of distinct interactions across all possible permutations is very large.

*Checkered Reshaping*: applies a reshape function for tensors that intertwines the features as seen in figure 4.9, which captures maximum heterogeneous interactions between entity and relation features.
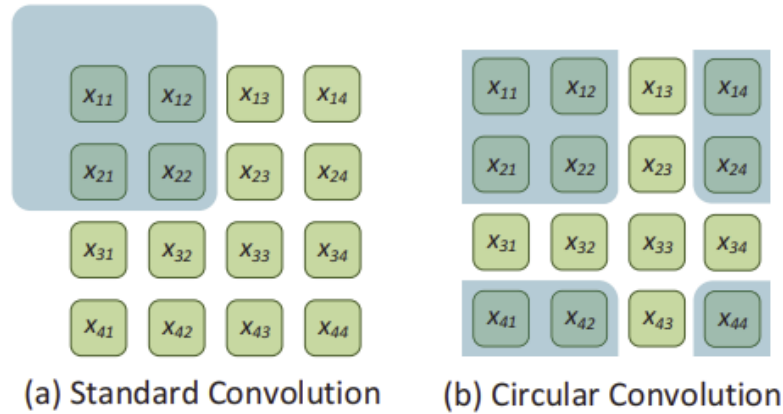


**Figure 4.10:** InteractE circular convolution. [100]

*Circular Convolution*: Circular convolution allows for a wraparound of the features selected for the convolution operation, as shown in figure 4.10.

Also improving on ConvE we have ParamE [9], which makes use of translational properties such as the models in section 4.2 and neural networks nonlinear fitting skills such as previous models from this section.

In ParamE, head entity, relation, and tail entity embeddings are both input and output of a neural network. The main idea of this model is to regard NN parameters
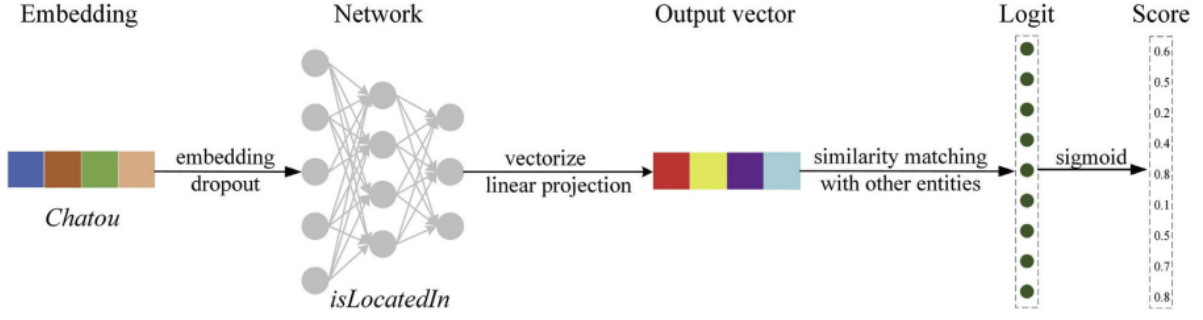
**Figure 4.11:** ParamE[9] model overview.

as relation embeddings, which makes them expressive and translational. Entity and relation embeddings in ParamE belong to feature space and parameter space respectively, which keeps them mapped into two different spaces in which the projections are performed and then matched further along the process.

ParamE's architecture has been tested with three different neural network configurations that use different kinds of layers: dense, convolutional, and gate structure ones. The latter is shown to outperform the others.

Finally, MEI [96] multi-partition embedding interaction model introduces the division of embeddings into multiple partitions, restricting the interactions of the embeddings of a triple (h,r,t) to only entries in the corresponding partitions of it, meaning the expressiveness of the most relevant elements of the embeddings is increased during training as they are restricted to each partition and are then multiplexed with other partitions, enhancing those particular set of features.
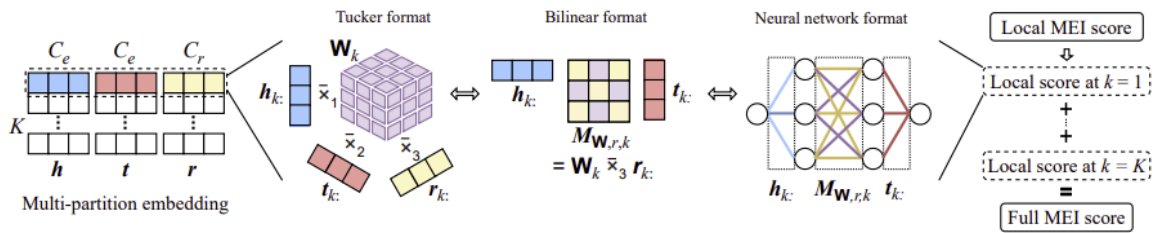


**Figure 4.12:** MEI model [96]

Interactions in each partition is done using by tensorizing the elements using the tucker format[98], then to learn the general linear interaction mechanisms they perform the tucker factorization operator producing a bilinear output which is then fed to the model's NN producing a score for that particular arrangement which is finally combined by performing the sum of all local interactions as described by figure 4.12.

In each triple (h, t, r), the entities and relations embedding vectors $h, t \in \mathbb{E}^{D_e}$, and $r \in \mathbb{R}^{D_r}$ are divided into multiple partitions, the score function of MEI, seen in Equation 4.8 is defined as the sum score of $\mathcal{K}$ local interactions, with each local

interaction being modeled by the Tucker format.

$$\mathcal{S}(h,r,t;\theta) = \mathbb{W}_k x_1 h_k x_2 t_k x_3 r_k \tag{4.8}$$

## 4.6 Summary

In this chapter, we have presented an overview of the existing methods that generate embedded representations of the information held in KGs. We have introduced and described the models that operate based on translation operations, to obtain numerical vectors that represent the information in KGs. We also described several techniques based on the semantic information of each node within the graph. As well as overviewing the methods based on representing the KG as a third-order graph factorization model and, finally, we have enumerated the methods that use neural networks to generate these embeddings.

# Chapter 5

# Reinforcement Learning

*"One learns from books and examples only that certain things can be done. Actual learning requires that you do those things."*

— Frank Herbert

Reinforcement Learning (RL) encompasses all techniques that train intelligent agents in a simulated environment via rewards to perform actions that modify said environment in some way. In this chapter, we get an overview of how Reinforcement Learning operates and the different algorithms and techniques that have arisen. Section 5.1 introduces Markov's Decision Process (MDP), which encompasses all RL problems. We also cover the elements found in every RL problem as well as how they interact with each other. Finally, we elucidate about the evolution of the different methods to perform Reinforcement Learning training. This chapter is sectioned as follows Section 5.2 showcases the evolution of RL algorithms throughout the years and reward implementations that became standardized. Section 5.3 shows some applications for Reinforcement Learning. Finally, Section 5.4 summarizes all the information given.

# 5.1 Introduction

Machine learning is a subfield of artificial intelligence that focuses on the development of algorithms and models that can learn from data. There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm is trained on a labeled dataset, where the input data is associated with a corresponding output label. The goal of supervised learning is to learn a mapping from input to output that can generalize to new, unseen data. In unsupervised learning, the algorithm is trained on an unlabeled dataset, where the goal is to discover patterns and structure in the data. Unsupervised learning can be used for tasks such as clustering[24, 70], dimensionality reduction[64, 109], and anomaly detection[34, 75].

Reinforcement learning is a type of machine learning that is used to solve problems where the agent interacts with an environment and receives feedback in the form of rewards. The goal of reinforcement learning is to learn a policy that maximizes the cumulative reward over time. Reinforcement learning is particularly useful for problems where the optimal action depends on the current state of the environment and the actions taken in the past. Reinforcement learning has been successfully applied to a wide range of problems, including robotics[23, 39, 47], game playing[77, 78, 101], recommendation systems[1, 69, 91, 106], and other areas.

Reinforcement learning is different from supervised and unsupervised learning in several ways. In supervised learning, the algorithm is provided with labeled data, whereas in reinforcement learning, the algorithm learns from feedback in the form of rewards. In unsupervised learning, the algorithm is trained on an unlabeled dataset, whereas in reinforcement learning, the algorithm interacts with an environment. Reinforcement learning also requires a different evaluation metric than supervised and unsupervised learning, since the goal is to maximize the cumulative reward over time rather than to minimize a loss function.

Reinforcement Learning techniques need to adapt to real-world environments, which is a challenging task given that reality is often diverse, non-stationary and open-ended. RL is generally described as having low generalization, i.e., it is hard to create general-purpose agents. This, in turn, generates another problem: environmental overfitting. Trained agents specialize in the environment in which they were trained and are very susceptible to small changes, needing to be re-trained very often when new knowledge is added[20, 112].

## Markov Decision Process

Markov decision processes (MDPs) are stochastic decision-making processes that use a mathematical framework as model. MDPs provide a formal framework for modeling

situations where outcomes are partly random and partly under the control of the decision-maker.

MDPs were introduced in the 1950s by Russian mathematician Andrey Markov who played a pivotal role in shaping stochastic processes. In their infancy, MDPs were used to solve business-related problems such as inventory management and control, queuing optimization, and routing matters. Today, MDPs find applications in studying optimization problems via dynamic programming, robotics, automatic control, economics or manufacturing.

In an MDP, the decision maker, or agent, interacts with an environment that is characterized by a set of states, actions, and rewards. The agent's goal is to learn a policy that maximizes the cumulative reward over time.
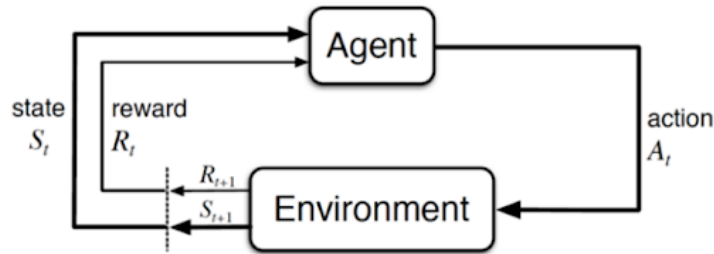


**Figure 5.1:** A Markov decision process loop.

The future state of the environment depends only on the current state and action, and not on the history of previous states and actions. This is known as the Markov property. This way the agent can use the current state of the environment to estimate the expected future rewards of different actions, and then choose the action that is most likely to lead to the highest cumulative reward over time.

Based on the notation from figure 5.1, the Markov property can be evaluated as shown in Equation 5.1.

$$P[St + 1|St] = P[St + 1|S1, S2, S3, ....., Sn] \tag{5.1}$$

Where the probability of the next state (P[St+1]) given the present state (St) is calculated based on all the possible next states (S1, S2, S3,..., Sn). The relation between the action to be chosen and the chosen outcome reward is referred to as the policy ($\pi$), $\pi : S-> A$.

To determine the best policy, it is essential to obtain optimal values produced for the agent's rewards at every state. The discount factor ($\gamma$) [0-1] determines which rewards are prioritized, minimal values focus on immediate rewards while maximum values focus on long-term rewards. The discounted infinite-horizon method expressed by the value function in Equation 5.2 determines the reward for each state by discounting future rewards.

$$V(s) = E \sum_{t=1}^{\inf} \gamma^t r_t | s_t \tag{5.2}$$

This value function can be separated into two components as shown in equations 5.3 and 5.4 which showcases the joint equation with both states and the split equation for the current and next states respectively.

$$V(s) = E[r + \gamma V(s_{t+1}) | s] \tag{5.3}$$

$$\Downarrow$$

$$V(s) = \mathcal{R}_s + \gamma \sum_{s_{t+1} \in S}^{\inf} \gamma^t r_t | s_t \tag{5.4}$$

or in other words, the reward for the current state and the value of the discounted reward for the next state. Furthering this line we arrive at Bellman's Equation (5.5).

$$V(s) = max_a [\mathcal{R}(s, a) + \gamma \sum_{s_{t+1} \in S} \mathcal{P}_{ss_{t+1}} V(s_{t+1})] \tag{5.5}$$

Where $s_{t+1}$ represents the next possible state, $s$ the current state, $\mathcal{P}_{ss_{t+1}}$ the probability of transitioning to $s_{t+1}$ based on the reward ($\mathcal{R}(s,a)$) for choosing action $a$.

The agent's actions and rewards vary based on the policy chosen for training, manifesting that the value function is different for each one. choosing the right policy is a type of operation that can be solved by iterative methods such as Monte Carlo evaluations, dynamic programming, or temporal-difference learning. The policy that selects the maximum optimal value while considering the present state is referred to as the optimal policy as seen in Equation 5.6

$$\pi(s) = argmax_a [\mathcal{R}(s, a) + \gamma \sum_{s_{t+1} \in S} \mathcal{P}_{ss_{t+1}}^a V(s_{t+1})] \tag{5.6}$$

The policy is, in essence, a particular application of Bellman's equation for each step, a new policy is evaluated based on the current state information.

## Reinforcement Elements

Reinforcement Learning can be defined as a Markov process that solves policy optimization functions by using Neural Networks for this reason RL problems are considered substrata of Markov's processes, and as such share a multitude of elements with them, for instance, the States, Actions, Environment, and Agent interactions

while also being respectful of Markov's Property. As Sutton and Barto put it, they are an "optimal control of incompletely-known Markov decision processes[88]".

Some core elements of RL problems are as follows:

- A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state.
- The agent also must have a goal or goals relating to the state of the environment.
- In supervised learning the agent learns from a labeled training set in order to extrapolate future answers not present on it. In RL agents must be able to learn from their own experience as when dealing with interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act.
- Reinforcement learning differs from unsupervised learning, one istries to maximize a reward signal instead while the other focuses on finding a hidden structure.

A key difference between RL and other types of learning is the exploration/exploitation dilemma: RL favors actions that provide high rewards (exploitation), but in order to reach the intended solution it may have to select low-reward actions in the early stages of training or actions that have not been selected before (exploration). The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best.

The elements of reinforcement Learning as we have previously commented are similar to MDPs. Sutton and Barto[88], apart from the environment and agent enumerate the following elements:

- The **policy** is the "brain" of the decision-maker in Markov's processes, an action mapping acquired from the perception of the agent from the current state of the environment that determines which action should be taken.
- The **reward signal** is the goal the Agent is trying to reach. Each action chosen by the Agent during training receives a numerical reward that must be maximized. It determines what constitutes a good or bad event while the agent learns and thus shapes its behavior. In general, reward signals may be stochastic functions of the state of the environment and the taken actions.
- The **value function** helps the agent determine what a good course of action to follow is in the long run. Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states
- Some RL systems rely on a **environment model**, which is a predictor for how the environment will behave, given a state and action the model can approximately

predict the next state and reward for that action. Some RL systems plan ahead and gather information about future states without actually experiencing them. these model-based methods contrast with the purely experience-based methods that originated RL.

## History

The roots of reinforcement learning can be traced back to early 20th-century psychologists like Thorndike[94] and B. F. Skinner[79, 80], who laid the groundwork for the concept of reinforcement by studying animal behavior.

In the 1950s and 1960s, control theory, which deals with the behavior of dynamic systems, influenced the development of reinforcement learning. Bellman's work on dynamic programming and optimal control provided a mathematical framework, introducing the Bellman equation as we previously discussed in equation 5.5.

Computer scientists in the mid-20th century explored ways to make machines learn from experience. A. L. Samuel's 1959 checkers-playing program, which employed temporal difference learning[72], demonstrated that machines could attain expertise in games without explicit programming. The 1980s saw the development of algorithms capable of learning from delayed rewards, including the widely-used Q-learning introduced by Watkins in 1989 [104].

The 1990s saw the convergence of reinforcement learning with neural networks, giving rise to deep reinforcement learning. Tesauro's TD-Gammon in 1995 showcased the potential of this combination by achieving expert-level play in backgammon[93]. In the 21st century, breakthroughs in deep reinforcement learning have been remarkable, exemplified by DeepMind's 2013 Deep Q-Network (DQN) algorithm[57] achieving superhuman levels of play in several Atari games.

From its origins in psychology to its current status as a powerful AI tool, the evolution of reinforcement learning has been marked by significant milestones and breakthroughs that are still ongoing. The following section will provide further details about the aforementioned proposals and study their evolution.

## 5.2   Algorithms and Techniques

This section will give an overview of each key evolution in RL development. As previously mentioned control theory is the origin of RL development, by evaluating and trying to solve dynamic systems, A. L. Samuel introduced Temporal Difference Learning [71] in a checkers game context. It is a type of reinforcement learning algorithm that updates the estimated value of a state based on the difference between the value of the next state and the current state. This difference is referred to as the TD-error.

Rather than aiming to compute the overall future reward on each step, the algorithm compares the new prediction with the previously stored one. If there is a disparity between the two predictions, the TD Learning algorithm quantifies this difference and employs it to update the previous prediction to the new one.

TD algorithms are based on Bellman equations. They compute optimal policies by recursively calculating the value of each state. This way the algorithm is able to learn from incomplete sequences of experience, where the final outcome is not known at the time of the update. This was a significant improvement in reward prediction and paved the way for more modern RL approaches.

Q-learning, developed by Watkins et. al. [104] in 1989, follows a similar methodology to that of TD learning, which updates the value of a state based on the difference between the value of the next state and the current state, Q-learning updates the value of a state based on the maximum value of the next state. The objective of Q-learning is to calculate an approximation $Q$ of the optimal action-value function $q^*$, which is independent of the policy, and store these updates in a table of action-value pairs called the Q-table. This makes Q-learning an off-policy algorithm, as it learns the optimal action-value function $Q$ regardless of the policy being followed. This action-value function is shown in Equation 5.7.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \qquad (5.7)$$

Where $\alpha \in (0, 1]$ is the step size, $S_t$ is the state in timestep t, $A_t$ the action in timestep t, $\gamma$ is the discount factor of rewards for the update, and $R_{t+1}$ the expected reward for the action chosen.

Even though Q-learning is an off-policy method, the policy still has an effect: it determines which states and actions to pick and evaluate for every episode. Q-learning works by iteratively updating the estimated value of a state-action pair according to the difference between the estimated value and the actual return obtained from taking that action in that state. The algorithm starts with an initial estimate of the action-value function and iteratively updates the estimates based on the observed returns. The algorithm continues to update the estimates until the action-value function converges to the optimal values as seen in figure 5.2.

One of the main advantages of Q-learning is that it can be used to learn optimal policies in environments with large or continuous state and action spaces. This is because Q-learning does not require a model of the environment, but rather learns from experience by iteratively updating the action-value function.

In the 1940s, John von Neumann and Stanislaw Ulam introduced the Monte Carlo method named after Monaco's famous casino, due to its reliance on randomness akin to a game of roulette.
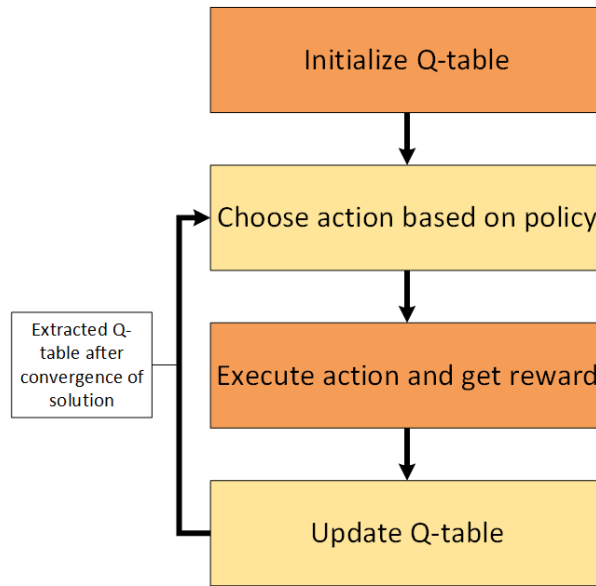
**Figure 5.2:** Q-learning operations diagram

Monte Carlo's methods estimate the value of a state or action based on the average return obtained by sampling complete episodes, unlike TD learning, which updates every step by comparison to previous results.

Monte Carlo methods use the actual return obtained from the episode to update the value of the states in that episode based on the actions taken, they are particularly useful in situations where the dynamics of the environment are unknown or stochastic, as they do not require a model of the environment, instead, they rely on experience to estimate the value of each action-state pair.
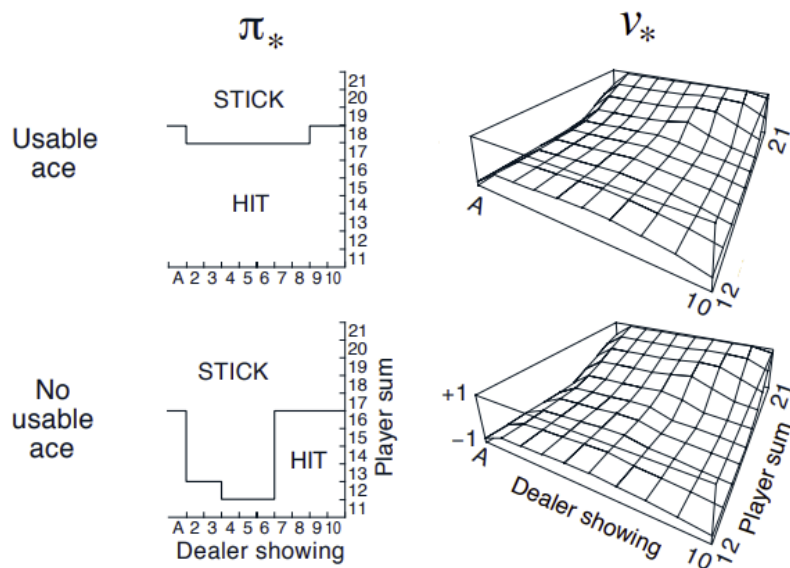


**Figure 5.3:** The game of blackjacks policy and value function as calculated by following a Monte Carlo control approach

Monte Carlo methods work by simulating complete episodes of interaction between the agent and the environment, starting from a given state and following a given policy (Off-policy Monte Carlo methods aside) this particular method of applying Monte Carlo to control theory is generally referred to as Monte Carlo control, Figure 5.3 shows the calculation of the policy and value function for a simplified game of blackjack.

The return obtained from each episode is then used to update the value of the states and actions that were visited for that episode. This process is repeated many times, with the value estimates converging to the true values as the number of episodes increases. Monte Carlo methods are unbiased and consistent, meaning that they converge to the true values of the states or actions with probability 1 as the number of episodes goes to infinity.

One of the main advantages of Monte Carlo methods is that they can be used to estimate the value of any state or action, regardless of its position in the state space or action space. This makes them particularly useful in large or continuous state and action spaces.

In contrast to the previous methods, the more modern Policy Gradient Methods differ from the previous approaches in that they do not focus on learning the values of actions to then select the highest one. Instead, Policy Gradient Methods learn a parameterized policy that can select actions without consulting a value function (equation 5.8). However, a value function may be applied to learn the policy ($\pi$) but is not required for action selection.

$$\pi(a|s,\theta) = Pr\{A_t = a | S_t = s, \theta_t = \theta\} \tag{5.8}$$

Where $\theta$ represents the policy parameters, and $a$ is the probability of that action being taken, in timestep t for the current state s.

This parametrized policy function takes the current state of the environment as input and outputs a probability distribution over possible actions. During training, the agent interacts with the environment, taking action according to its current policy. The rewards obtained from these actions are used to adjust the parameters of the policy function through a process called gradient ascent, this process is as follows:

- If an action leads to a higher reward, the probability of taking that action in similar future situations is increased.
- If an action results in a lower reward, the probability of taking that action is decreased.
- This adjustment is guided by the gradient of the expected cumulative reward with respect to the policy parameters.
- Iteratively updating the policy based on this gradient, the agent learns to make better decisions over time, improving its performance in the given environment.

Policy gradient methods learn the policy parameter $\theta$ according to the gradient of some performance measure $J(\theta)$ linked to the policy parameter and seek to maximize this performance measure as depicted in equation 5.9.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{5.9}$$

Where $\nabla J(\theta_t)$ is an approximation of the gradient of performance $J(\theta)$ for the timestep t and $\alpha$ is the learning rate a parameter to control the speed of the agent learning. All methods that follow this methodology of approximating a gradient for performance linked to a parametrized policy are what encompasses policy gradient methods. They offer flexibility and can handle high-dimensional action spaces, making them well-suited for complex tasks. However, training can be computationally intensive, requiring multiple interactions with the environment to fine-tune the policy.

The action selection in policy gradients is a stochastic event, where the probability of an action is computed as the estimated reward for the state that action leads to, making the process a stochastic gradient ascent.

REINFORCE [89] is the first policy-gradient-based algorithm that follows this stochastic gradient ascent method. The main idea behind it is to maximize the expected cumulative reward by adjusting the parameters of the policy $\theta$ in the direction that increases the likelihood of actions leading to higher rewards so that it learns to take them. In other words, it maximizes the policy gradient function $\nabla J(\theta_t)$ by finding the optimal $\theta$ for the maximization of $J(\theta)$, formally defined by Equation 5.10.

$$\nabla_\theta J(\theta) = \mathcal{E}_\pi [\sum_{t=0}^{inf} \nabla_\theta log \pi(a_t|s_t; \theta) G_t] \tag{5.10}$$

$G_t$ is the return at timestep t, identified by Equation 5.11

$$G_t = \sum_{k=0}^{inf} \gamma^k R_{t+k+1} \tag{5.11}$$

$\nabla_\theta log \pi(a_t|s_t; \theta)$ is the score function representing the gradient of the log probability of taking action $a_t$ in state $s_t$ with respect to the policy parameters $\theta$, REINFORCE then updates the policy parameter and performs stochastic gradient ascent as seen in previous Equation 5.9. The key intuition is that the policy parameters are adjusted in proportion to the gradient of the expected return, encouraging actions that lead to higher rewards to become more likely.

REINFORCE shares similarities with Monte Carlo processes, particularly in how it estimates expected values. In both REINFORCE and Monte Carlo methods, an agent interacts with the environment over a sequence of steps and receives a cumulative

reward at the end of each episode. Instead of directly calculating the expected cumulative reward, both approaches estimate it by sampling multiple trajectories (sequences of states, actions, and rewards) and averaging the observed returns.

REINFORCE, in comparison to traditional Monte Carlo methods, offers the key improvement of leveraging policy gradients, enabling the algorithm to handle continuous action spaces and apply parametrized policies, often implemented with neural networks. The probabilistic nature of REINFORCE facilitates effective exploration strategies, reducing variance in estimated returns and promoting stable convergence during training.

Actor-critic methods represent a class of reinforcement learning algorithms that combine elements of both policy-based (actor) and value-based (critic) approaches. The fundamental idea is to have two distinct components within the learning agent: an **actor** that learns a policy to select actions, and a **critic** that evaluates the value of state-action pairs. This dual structure allows for more efficient learning and improved stability compared to pure policy or value-based methods.

The actor update function uses the policy gradient of the expected cumulative reward with respect to the policy parameters, which we have previously explored in REINFORCE (eq. 5.10) without the expected return $G_t$, and instead relying on an advantages function $Q(s, a)$ that represents the difference between the estimated value of a state-action pair and the value function only in the evaluated state as shown in equation 5.12.

$$\nabla_\theta J(\theta) \approx \mathcal{E}_\pi[\nabla_\theta log \pi(a_t|s_t; \theta) Q(s, a)] \tag{5.12}$$

The critic parameters $(w)$, and $\beta$, the critic's learning rate, are updated using Temporal Difference (TD) learning methods. the critic tries to approximate the TD error $\delta$ between the estimated value $V(s_t; (w))$ and the current cumulative reward $R_{t+1}$ plus the discounted value of the next state $\gamma V(s_{t+1}; (w))$, formally described by equation 5.13.

$$\delta = R_{t+1} + \gamma V(s_{t+1}; (w)) - V(s_t; (w)) \tag{5.13}$$

The TD error measures the difference between the agent's prediction of the immediate reward and its estimate of the future rewards, providing a signal for updating the critic's parameters, which are updated by using the gradient of the TD-error with respect to the critic's parameters $(w)$ according to the learning rate $\beta$, formally described by equation 5.14.

$$(w)_{t+1} = (w)_t + \beta \nabla_{(w)} \delta V(s_t; (w)) \tag{5.14}$$

This equation reflects the iterative adjustment of the critic's parameters to minimize the TD error, aligning the predicted values with the observed rewards. The learning rate ($\beta$) determines the step size in the parameter space.

Actor-critic methods strike a balance between exploration (through the actor's policy) and exploitation (through the critic's value estimates), reducing the variance caused by policy gradient Monte Carlo methods.

## 5.3   Applications

This section provides a concise exploration of diverse applications harnessing the power of reinforcement learning (RL). From gaming to Autonomous driving or Industrial control, RL exhibits its versatility by optimizing decision-making processes across an array of domains.

### Games

Whether in traditional board games or cutting-edge video games, reinforcement learning has demonstrated remarkable prowess in mastering strategic decision-making and optimizing gameplay. These approaches showcase the evolution of AI's gaming capabilities and their broader implications.

AlphaGo[77] represents a groundbreaking application of reinforcement learning in the realm of board games, specifically Go. Developed by DeepMind, AlphaGo employs a combination of deep neural networks and Monte Carlo Tree Search to navigate the immense complexity of the game. Its neural network-based policy and value functions enable strategic decision-making, while the Monte Carlo Tree Search facilitates lookahead exploration. Notably, AlphaGo's triumph over human champion Fan Hui marks a turning point in machine learning history, showcasing the capability of reinforcement learning to master complex strategic domains with profound implications for artificial intelligence and game theory research.

AlphaZero[78] extends the paradigm established by AlphaGo, utilizing a reinforcement learning approach for mastery in a broader spectrum of board games, including chess and shogi. AlphaZero relies solely on self-play reinforcement learning without access to human game data or domain-specific knowledge. Its unparalleled success in surpassing world-champion level performance across multiple games displays the ability of RL to achieve superhuman proficiency and generalization within strategic gaming domains.

However, AlphaStar[101] marks a notable advancement in the application of reinforcement learning, applying similar techniques for real-time strategy games. Leveraging a combination of deep neural networks and a unique multi-agent training setup, AlphaStar achieves superhuman proficiency in StarCraft II, surpassing top

human players. Its capacity to handle the complexity of real-time decision-making within the game environment underscores the robustness and scalability of reinforcement learning methodologies in addressing challenges beyond turn-based or board game scenarios.

## Industrial Control

But Reinforcement learning can be applied to much more than games, it proves its versatility in addressing real-world challenges, for example, in the world of controlling operations in industrial processes.

A prime example would be data center cooling, by enhancing safety and efficiency within critical infrastructure settings, traditional control systems often struggle to adapt dynamically to complex and changing environments, leading to inefficiencies and potential safety risks.

Applying RL techniques to the parameter optimization problem that is expressed by the cooling system it can result in significant energy savings and reducing CO2 emissions to help combat climate change.

The approach works by polling the server sensors continuously and feeding the information given by these sensors to deep neural networks, which predict how different actions will affect future energy consumption. The system identifies how to minimize energy consumption and maintain satisfactory safety constraints. Those actions are sent back to the data center, where the actions are verified by the local control system and then implemented.

Other techniques in industrial control came with new interests in Nuclear fusion, new tokamak designs in toroidal shapes have been produced and Reinforcement Learning provided a way to optimize the magnetic polarity to maintain the temperature inside these machines [17].

RL has produced and controlled a diverse set of plasma configurations for the Tokamak and achieves accurate tracking of the location, current and shape of these configurations. This represents a notable advance for tokamak feedback control, showing the potential of reinforcement learning to accelerate research in the fusion domain, and is one of the most challenging real-world systems to which reinforcement learning has been applied.

## Autonomous Driving

Autonomous driving presents a transformative shift in transportation that holds immense potential to enhance our lives. The promise of significantly improved safety on the roads, with autonomous vehicles minimizing human errors and reducing traffic accidents.

Beyond safety, autonomous driving has the capacity to enhance accessibility for individuals facing transportation challenges, fostering independence and inclusivity. The technology's potential to optimize traffic flow promises more efficient and less congested road networks, translating to reduced travel times and environmental impact.

Moreover, the prospect of increased productivity and improved quality of life emerges as individuals can utilize travel time for work or relaxation. The evolution of urban planning and space utilization, prompted by autonomous driving, may lead to more sustainable and pedestrian-friendly cities. While challenges persist, the multifaceted benefits of autonomous driving suggest a future where transportation becomes safer, more efficient, and enriching for individuals and communities alike.

Multiple methods exist that leverage reinforcement Learning for controlling vehicles automatically, they approach multiple tasks such as Intent prediction for traffic actors such as pedestrians or other vehicles, path planning and trajectory optimization, development of high-level driving policies for complex navigation tasks, scenario-based policy learning for highways, merges and splits, intersections, motion planning and dynamic path planning, controller optimization, learning of policies that ensures safety and perform risk estimation[38].

## 5.4   Summary

The contents of this chapter have provided an overview of the methods improvements and applications of reinforcement learning in multiple contexts, We first learned about Markov processes and how RL fell under this umbrella, the elements that compose a reinforcement learning problem and the history of the methods that improved general RL. Then we learned about the algorithms at the core of RL and how each one improved upon the shortcomings of the previous ones. Finally, we observed some applications of RL to real-world problems.

# Part III

# Our Proposal

# SpaceRL: Our Knowledge graph reasoning proposal.

*"Reason has always existed, but not always in a reasonable form."*

— Karl Marx

When presented with the challenge of completing a given knowledge graph we must aim to offer the most complete answer possible, a reasoned answer that complements the inference achieved, that responds to the question of why the knowledge presented must be incorporated back into the graph. In this chapter, we introduce SpaceRL our knowledge graphs reasoning proposal. This chapter is structured as follows: Section 6.1 introduces the proposal , Section 6.2 Formally introduces the problem to solve., Section 6.3 Presents the developments made in the work, Section 6.4 Explains the methodology followed to evaluate the methods and displays the results obtained by them, Section 6.5 touches on the limitations of the method, Lastly, Section 6.6 summarizes the work.

## 6.1 Introduction

This chapter introduces SpaceRL-KG our KG reasoning proposal using Reinforcement Learning with embedding-based rewards.
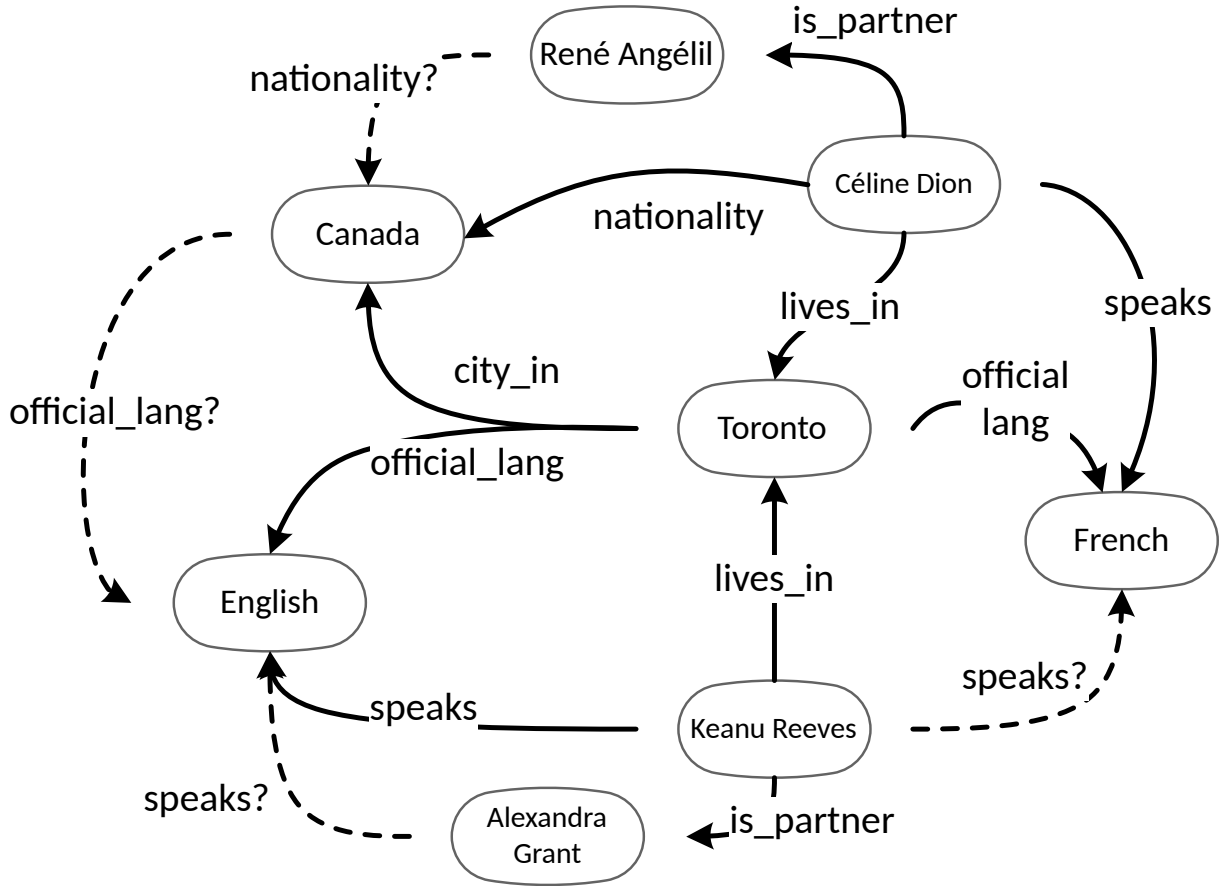


**Figure 6.1:** An example Knowledge Graph with possible new relations as dotted lines.

To apply RL to Knowledge Graph Completion, we train a Reinforcement Learning agent that learns how to navigate through the graph and generate new reasoned paths that can be applied back as new triples.

Using the graph from Figure 6.1 as an example we will overview how the agent would infer the new knowledge ("Alexandra Grant", "speaks", "English").

The agent would start at the subject node ("Alexandra Grant") and decide what edges should be traversed to reach the target node ("English"), these edges are represented as triples ($node_0$, $relation$, $node_1$) indicating directionality. In this case, the agent could deduce that Alexandra Grant speaks English by navigating through several paths, such as:

(Alexandra Grant, ¬is_partner, Keanu Reeves) →
(Keanu Reeves, speaks, English)

(Alexandra Grant, ¬is_partner, Keanu Reeves) →

$$\text{(Keanu Reeves, lives\_in, Toronto)} \rightarrow$$
$$\text{(Toronto, official\_lang, English)}$$

The relational paths are then registered as possible answers to the query ("Alexandra Grant", "speaks", "?") where the sum of the relations of the path would equal the missing target relation "speaks", in this case, the relational paths are the following.

$$(e_0, \neg\text{is\_partner}, e_1, \text{lives\_in}, e_2, \text{official\_lang}, e_q) = (e_0, \text{speaks}, e_q)$$
$$(e_0, \neg\text{is\_partner}, e_1, \text{speaks}, e_q) = (e_0, \text{speaks}, e_q)$$

In this example it is confirmed by two separate paths that Alexandra Grant speaks English, the more this relation chain appears in the graph, the more trustworthy it becomes. These reasoned paths can be directly translated into new graph triples or presented to users for human-in-the-loop relation classification operations.

The application of RL to KG reasoning tasks requires that the entities and edges of the graph be transformed into numerical vectors that can be provided to the agents as a representation of the state and its available actions at every step.

SpaceRL-KG focuses on an improved set of rewards and the application of novel algorithms to these processes. The evaluation of our technique is performed by applying it to several widely accepted Knowledge Graphs and shows that our novel reward functions significantly improve performance when compared to more traditional ones, especially when node embeddings are used. Throughout this chapter, we will expand on the running example in Section 6.1.

## 6.2 Formal Description

> se ha separado cada parte de la estructura formal del problema y solucion en los diferentes apartados y este se ha reservado tan solo para la descripcion formal del problema.

Knowledge graphs $\mathcal{K}$ are formally represented as a set of entities $\mathcal{E}$ and relations $\mathcal{R}$ Where $(s, r, o)$ denotes a triple in the graph, representing a fact that connects entity $s$ to $o$ via a relation $r$ formally described by equation 6.1.

$$\mathcal{K} = \{(s, r, o) \mid s, o \in \mathcal{E}, r \in \mathcal{R}\} \tag{6.1}$$

We assume that, due to its nature, $\mathcal{K}$ is inherently incomplete concerning the information that we know to be true in the real world.

In order to find missing edges in these Knowledge Graphs we perform a multi-hop approach where the input is a query represented by a source node and a relation

$(e_0, r_q)$ and the output is a path of predetermined length $n$ that should reach the answer node of the missing edge, $e_a$. This path can be represented as:

$$p_k(e_0, e_k) = \left\{ e_0 \xrightarrow{r_0} e_1 \xrightarrow{r_1} ... \xrightarrow{r_n} e_n \right\} \tag{6.2}$$

And it is considered to be correct if $e_n = e_a$, where $e_n$ is the node in step n, the last step in the path. We could then infer a new relation $(e_0 \xrightarrow{r_q} e_n)$.

## 6.3  Our proposal

As we have already seen in previous sections SpaceRL-KG focuses on acquiring reasoned knowledge in the form of paths in order to determine if a particular relation should exist between two nodes of the graph. It does so by training intelligent agents by way of applying novel reward functions and algorithms in a Reinforcement Learning setup, where the Knowledge Graph acts as the environment and each node as the state.

To obtain these paths the graph nodes and relations must be encoded as numerical vectors representing their position in a N-dimensional space. These numerical vectors are then combined in a way where they represent the current state, a possible action and the context of the episode. this information is then fed to a Neural Network which represents the agent policy, responsible for selecting the best-estimated action for every episode step.

This is done by evaluating every possible action for every step of the path and then stochastically selecting one based on the policy scores provided, the agent is then rewarded based on several metrics and updated after every episode.

### 6.3.1  Embedding representations

### 6.3.2  Reinforcement Learning implementation

Any application of Reinforcement Learning to knowledge graphs can be formalized as a Markov decision process, assuming that several conditions are met beforehand. By adding inverse edges to the graph, path connectivity is guaranteed: if the edge $(e_i, r, e_{i+1})$ exists in the graph, so does $(e_{i+1}, \neg r, e_i)$, where $\neg r$ denotes the inverse relation to $r$. During the training of the model, we remove the edges that are used to create queries in order to simulate the absence of their direct answers and prevent the agent from taking the direct path to the target entity. Self-loop edges are also added beforehand; these represent the NO-OP action, which entails that an agent chooses to stay in the current node, which is desirable if the answer node is reached before advancing $n$ steps. This introduces a new edge $(e_i, r_{NO-OP}, e_i)$ per entity. Staying in

the current node might cause local minimum stagnation, which is undesired behavior and should be accounted for by the reward function.

We can define the following elements typical of a Markov process that, in turn, make up the agent and the environment:

**State:** The state $S_t$ at a certain step $t \in 0..n$ is defined as the combination of the query $(e_0, r_q)$, the destination node $e_n$ and the current location $e_t$. We can formally define the state as follows: $S_t \in \mathcal{S} \parallel S_t = (e_t, e_0, r_q, e_a)$ where $\mathcal{S}$ is the set of all possible states.

**Observations:** The environment is not fully observed by the agent: in any given state $S_t$ the agent is only aware of its current location $e_t$ and the query $(e_0, r_q)$. Formally, $O_t \in \mathcal{O} \parallel O_t = (e_t, e_0, r_q)$ where $\mathcal{O}$ is the set of all possible agent observations.

**Actions:** The set of all actions an agent can take in a given state $t$ depends on the current location, $e_t$. Since the location represents a node in the graph with one or more connected edges, these edges are the possible actions of the agent. Formally we describe the action space for a given location $e_t$ with

$$\mathcal{A}_t = \left\{ (e_t, r_i, e_i) \parallel S_t = (e_t, e_0, r, e_a), r_i \in \mathcal{R}_i, e_i \in \mathcal{E}_i \right\} \tag{6.3}$$

where $\mathcal{R}_i, \mathcal{E}_i$ denote the subset of relations and entities from the edges connected to $e_t$ respectively. In simpler terms the agent can select any of the outgoing edges, being aware of the node it reaches.

**Transition:** The transition refers to how the environment evolves after the agent takes an action. In this case this happens in a deterministic manner as previously mentioned, updating the set of available actions to those of the new node that the agent reached through the selected action. The new state remains the same except for the location, which becomes the node reached by the chosen action. Formally, this is represented by a transition function involving the state and the action space such as $\mathcal{P}(\mathcal{S}, \mathcal{A}) = \mathcal{P}(e_t, (e_0, r_q), \mathcal{A}_t))$, where $(e_0, r_q)$ denotes the query, $e_t$ the current location, and $\mathcal{A}_t$ the action space for that location.

**Rewards:** For the computation of the reward, we have defined several components that can be toggled or applied with lesser or greater weights:

- **Terminal**: a binary {1,0} reward that is given to the agent whenever it is located in the answer node $e_a$. If triggered, this reward overrides all other reward components.
- **Distance**: a reward component based on the distance to the answer node (length of the shortest path). Shorter distances lead to higher rewards.
- **Embedding**: a reward component based on several properties of the embedding representation of nodes and relations, representing semantic similarity. Higher

semantic similarity leads to higher rewards.

- **Shaped**: a variant of the terminal reward function that replaces the 0 score associated with not reaching the target node with an embedding-based score computed from the starting node, the relation representing the question, and the reached node. Note that this reward does not compare the reached node with the target one, but uses traditional embedding-based score as a fallback function.

- **NO-OP**: a negative reward that aims to discourage the use of the no-op action outside the answer node. This reward overrides all other reward components, and cannot be triggered in the same step as the terminal reward.

Detailed information regarding the distance and embedding rewards can be found in Section

Shaped reward were implemented as seen in approaches for comparison purposes; they use embeddings, however, they do so in a different manner, only using them as an anticipation or guiding factor and relying in terminal rewards.
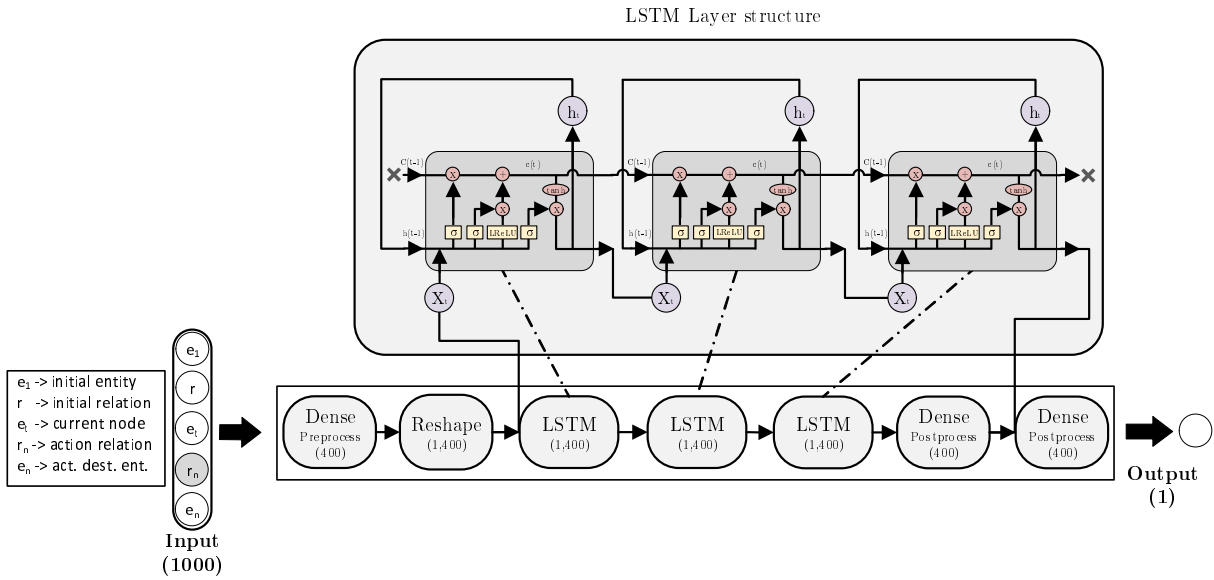


**Figure 6.2:** Policy architecture.

### 6.3.3 Policy Network

The policy used to determine which action will be taken at each step is based on the neural network architecture described in Figure 6.2. The network receives state information in the form of the query $(e_0, r_q)$, the current location node $e_t$ as well as the action which is being evaluated $(r_{t+1}, e_{t+1})$. These actions are all the possible relations connected to the current node $e_t$. It is necessary to evaluate each action individually since the number of edges connected to a node is variable.

For the neural network to receive this information, it is encoded using translation embedding representations for each of the entities and relations so that $\mathbf{r} \in \mathbb{R}$, $\mathbf{e} \in \mathbb{E}$

where $\mathbb{R}$ and $\mathbb{E}$ are the vectorial spaces containing all possible representations in the graph. The chosen size for these embedded representations was 200 as it is a common in literature for these set of knowledge graphs and embedding combinations, making the networks input size 1000 units.

The embedding vectors are concatenated and fed to the network as $[e_q, r_q, e_t, r_{t+1}, e_{t+1}]$. The input is passed on to a densely connected layer for pre-processing, whose output is connected to a multi-layered Long short-term memory (LSTM) block with 3 layers. The result is post-processed with 2 more dense layers that produce a numerical answer representing the quality of the selected action. The final output is produced by a sigmoid function( 6.4) that produces a [0,1] output that is adequate for an action evaluator.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{6.4}$$

The activation function in intermediate layers was chosen to be leaky ReLU to avoid gradient explosion , which is common in deep neural approaches. Batch normalization is applied to the embedding representations, and ADAM is used as optimizer.

The historical values $h_t = (e_0, r_1, e1, ..., r_n, e_n)$ can be formally defined by equation( 6.5) for the hidden state layer of LSTMs at instant t.

$$h_t = LSTM_{enc}((r_t, e_t), h_{t-1}) \tag{6.5}$$

Finally the policy network can be formalized as:

$$\pi_s(a_t|s_t) = \sigma(\mathcal{A}_t \times \mathcal{S}_t[W_6; W_5] ReLU([W_4 - W_2] \times [e_t; r_t; h_t] ReLU(W_1[e_t; r_t]))) \tag{6.6}$$

### 6.3.4 Embedding & Distance rewards

These reward functions act on a step by step basis, that is, they are calculated at every step independently of whether or not the answer node was reached. The embedding reward is based on the property of translational embeddings according to which the combination of an entity and a connecting relation results in a very similar vector to the target entity. We leverage this information to define a new reward component, as shown in Algorithm 1. This function intends to reward the semantic similarity between the reached node and the answer node. This is done by measuring the distance between their embedding representations using several distance functions and checking whether such distance has decreased with respect to the former step.

The distance reward is computed according to the length of the shortest path from

---

**Algorithm 1:** Embedding-based reward calculation

---

**Input:**
// Previous location values.
*PrevDot, PrevEucDist, PrevCosSim*:<Float>
// Embedding of the current node
$e_t$: List<Float>
// Embedding of the goal node
$e_a$: List<Float>
**Output:** *reward*: Float

1   $dot \leftarrow e_t \cdot e_a$ // "·" is the dot product operator.
2   $euc\_dist \leftarrow norm(e_t - e_a)$
3   $cos\_sim \leftarrow dot \;/\; (norm(e_t) \times norm(e_a))$
4   $reward \leftarrow 0.0$
5   **if** *PrevDot* > *dot* **then**
6     |   $reward \mathrel{+}= \frac{1}{3}$
7   **else if** *PrevDot* = *dot* **then**
8     |   $reward \mathrel{+}= \frac{1}{6}$
9   **if** *PrevEucDist* > *euc_dist* **then**
10     |   $reward \mathrel{+}= \frac{1}{3}$
11   **else if** *PrevEucDist* = *euc_dist* **then**
12     |   $reward \mathrel{+}= \frac{1}{6}$
13   **if** *PrevCosSim* > *cos_sim* **then**
14     |   $reward \mathrel{+}= \frac{1}{3}$
15   **else if** *PrevCosSim* = *cos_sim* **then**
16     |   $reward \mathrel{+}= \frac{1}{6}$
17   **return** *reward*

---

---

**Algorithm 2:** Distance reward calculation

---

**Input:**
// Current distance to goal node
$d_t$: Integer
// Previous distance to goal node
$d^{-1t}$: Integer
**Output:** *reward*: Float

1   $reward \leftarrow 0.0$
2   **if** $d_t < d^{-1t}$ **then**
3     |   $reward \leftarrow 1$
4   **else if** $d_t = d^{-1t}$ **then**
5     |   $reward \leftarrow \frac{1}{3}$
6   **return** *reward*

---

the currently explored node $e_t$ to the answer node $e_a$. The distance is calculated by performing a tree exploration of the graph, which is an expensive operation. This is why we opted for caching computed distances in order to share these data between training episodes. The calculated distance *d* is then compared against the previously

computed distance $d^{-1}$ at $e_{t-1}$ to check if the chosen action got the agent closer to the end node. This way, the agent is rewarded by moving faster towards the answer node, which results in shorter paths without the need of limiting their length excessively by reducing $n$. The distance reward function is shown in detail in Algorithm 2.

### 6.3.5  Reinforcement Learning algorithms

re-wrie this without making references to the table of SoTa approaches.

$$\Delta_\theta \mathcal{J}(\theta) = \Delta_\theta log \pi_\theta(s,a)\mathcal{G}(s,a) \tag{6.7}$$

where $\mathcal{J}$ can be any loss function and $\mathcal{G}$ is the propagated reward in the action state pair (a,s).

This approach, however, suffers from some known shortcomings:

- Since the agent takes several actions during the episode, specially if several episodes take place during the same learning loop for faster training times, the variance of the method increases (that is, a single reward is used to influence many different actions), and it becomes less likely to assign proper credit to the actually useful actions. Because of this, it takes more time for the gradients of the agent to converge, increasing the training time needed to stabilize the value of the loss function.
- REINFORCE only produces training data when episodes conclude, meaning that it is not possible to train an agent with a single, longer episode. This, however, has no effect in our context since all episodes are limited to a number of steps N.
- REINFORCE is very sensitive to hyperparameters, making it crucial to tune them for each approach, as seen in MINERVA , where a hyperparameters table is specified for the performed tasks.

To overcome REINFORCE's limitations, several other paradigms have been developed. We have identified as particularly promising the Proximal Policy Optimization (PPO) and Actor-Critic approaches, in particular, the soft variant These algorithms mitigate the aforementioned problems. Their combination benefits from the advantages typical of Temporal-Difference Learning by defining a critic: a neural network tasked with predicting the value of the long-term reward associated to an action-state pair $(a,s)$. This value is used in a similar fashion to how Q-learning uses the Q-value in its learning process. The values provided by the critic $\mathcal{C}(s,a)$ replace $\mathcal{G}(s,a)$ in equation 6.7, which results in the following equation:

$$\Delta_\theta \mathcal{J}(\theta) = \Delta_\theta log \pi_\theta(s,a)\mathcal{C}(s,a) \tag{6.8}$$

We then learn the equivalent Q-value through the aforementioned TD-learning techniques, formally described as:

$$C(s,a) \leftarrow Q(s,a) + \alpha(r(s,a) + max_{a+1}\gamma Q(s^{-1},a^{-1}) - Q(s,a)) \qquad (6.9)$$

where $a+1$ denotes the possible action values in the next step, $\alpha$ and $\gamma$ are numerical hyperparameters, and $Q(s,a)$ denotes the output given by the actor from the action value pair $(s,a)$.

This approach reduces the variance significantly as we can update the parameters in a step by step basis by using the Q-value-like method, ensuring faster convergence of the policy gradient and making it possible to run a non-episodic method.

## 6.4 Evaluation

In this section, we describe in detail the experiments we carried out to evaluate our contributions and their results. First, we introduce the datasets we use in our experiments. Then, we describe the experimental setup, that is, the techniques that are evaluated and under what conditions. Finally, we provide a discussion of the results obtained in our experiments.

### 6.4.1 Experimental data

We selected five benchmark datasets, which are often used in the literature, and which can help discern the strengths of the techniques under evaluation:

- **COUNTRIES** , a small, low-connectivity dataset that contains relations between geographical regions and the countries inside them.
- **Unified Medical Language System (UMLS)** , a highly connected dataset that consists of biomedical data, with entities representing different diseases, bacteria, treatments and diagnoses.
- **FreeBase (FB15K-237)** , a subset of the FreeBase Knowledge Graph in which inverse relations have been removed to avoid leakage from the training to testing validation splits.
- **Never Ending Language Learning (NELL-995)** , a particular version of the NELL dataset, built by web crawlers automatically by extracting triples from plain text from several sources.
- **WordNet (WN18RR)** , a subset of WordNet in which originally many text triples are obtained by inverting triples from the training set. This leads to these datasets being able to be completed by using simple ruling, by removing these inverse

triples, WN18RR, avoids leakage of inverse relations into the testing split, thus making it necessary to have knowledge of the entire dataset.

Table 6.1 contains a statistical summary of the aforementioned datasets.

| Dataset | Entities | Relations | Triples | Degree Mean | Degree Median |
|---------|----------|-----------|---------|------|--------|
| COUNTRIES | 272 | 2 | 1,159 | 4.35 | 4 |
| UMLS | 135 | 49 | 5,216 | 38.63 | 28 |
| FB15K-237 | 14,505 | 237 | 272,115 | 19.74 | 14 |
| NELL-995 | 75,492 | 200 | 154,213 | 4.07 | 1 |
| WN18RR | 40,945 | 11 | 86,835 | 2.19 | 2 |

**Table 6.1:** Datasets (Degree = degree of connectivity)

## 6.4.2   Experimental setup

| Metric | Embedding | Evaluation COUNTRIES | UMLS | C-Normal | U-Normal |
|--------|-----------|-----------|------|----------|----------|
| HITS @1 | **TransE** | 0.819 | 0.298 | **1.000** | **1.000** |
| | DistMult | 0.799 | 0.271 | 0.732 | 0.693 |
| | ComplEx | 0.743 | 0.284 | 0.000 | 0.845 |
| | TransR | 0.795 | 0.212 | 0.685 | 0.000 |
| HITS @3 | **TransE** | 0.996 | 0.656 | **1.000** | **1.000** |
| | DistMult | 0.991 | 0.604 | 0.676 | 0.628 |
| | ComplEx | 0.981 | 0.646 | 0.000 | 0.927 |
| | TransR | 0.992 | 0.517 | 0.716 | 0.000 |
| HITS @5 | **TransE** | 1.000 | 0.832 | 0.000 | **1.000** |
| | DistMult | 1.000 | 0.787 | 0.000 | 0.632 |
| | ComplEx | 0.999 | 0.823 | 0.000 | 0.925 |
| | TransR | 0.999 | 0.711 | **1.000** | 0.000 |
| HITS @10 | **TransE** | 1.000 | 0.973 | **1.000** | **1.000** |
| | DistMult | 1.000 | 0.956 | 1.000 | 0.704 |
| | ComplEx | 1.000 | 0.969 | 1.000 | 0.926 |
| | TransR | 1.000 | 0.916 | 1.000 | 0.000 |
| MRR | **TransE** | 0.904 | 0.520 | **1.000** | **1.000** |
| | DistMult | 0.891 | 0.480 | 0.733 | 0.618 |
| | ComplEx | 0.856 | 0.502 | 0.000 | 0.831 |
| | TransR | 0.890 | 0.415 | 0.693 | 0.000 |

**Table 6.2:** Embedding comparison for the UMLS and COUNTRIES datasets

First, in order to limit our experimentation to a single type of embedding, we carried out a preliminary experimental study to determine which embeddings offered the best results. The results of this study are shown in Table 6.2. It can be observed that TransE, despite its simplicity, leads to the best results in the COUNTRIES and

UMLS datasets. Consequently, we use TransE for all of our experiments over the other three types that were tested (DistMult, ComplEx and TransR).

Previous to every training loop we make sure that the vectorial space of the embeddings is normalized, to avoid vanishing gradients or gradient explosions as previously mentioned. We set the hyperparameters to the following values: $\alpha = 0.9$, $\gamma = 0.99$, path exploration to a maximum of 5, and hidden size of intermediate layers to 400. These values were selected on an empirical basis, as to the best of our knowledge there is no way to systematically find the optimal ones for a given architecture. For the Policy network, we set the kernel initializer to Glorot/Xavier for general purpose or LeCun for incompatible activation functions. We chose LReLU with a factor of 0.01 as the activation for the intermediate layers to prevent gradient vanishing, which would lead to the Glorot activation being prevalent. We used L1 and L2 regularisation with L1 having a factor of $10^{-5}$ and L2 having $10^{-4}$. We selected RSMprop as an optimiser for the PPO network and Adam as default. The previous choices, were based on empirical testing performed by previous approaches

We performed a preliminary evaluation of several techniques to select the most promising ones for further experimentation:

- **Simple Terminal:** A non-retropropagated training with terminal reward and direct transmission of rewards, the simplest algorithm possible, that serves as a baseline.
- **Retropropagated Terminal:** A REINFORCE algorithm training with just a terminal reward. This strategy represents the current trends in the state-of-the-art.
- **Simple Embedding:** An embedding-based reward training with a simple algorithm.
- **Simple Distance:** A distance reward training with a simple algorithm.
- **Simple Combined:** A combinational reward training which uses both the embedding rewards (70%) and the distance rewards (30%), and a simple algorithm. The factor for each of the rewards is based on their previous performance when used individually.
- **Simple Distance (PPO):** A distance reward training with the PPO algorithm.
- **Simple Embedding (PPO):** An embedding reward training with the PPO algorithm.

We generate a test and a training set from the triples in the dataset by following a 1-to-7 rotary rule, where 7/8 of the total set would be selected at random and the remainder 1/8 would be used as the test set. We evaluate the quality of the generated model by using two widely accepted performance measures: Mean Reciprocal Rank (MRR) of all correct entities, which is calculated as the mean of $1/rank_e$, with $rank_e$ being the ranking position of the target entity, and Hits@N, which is the percentage of
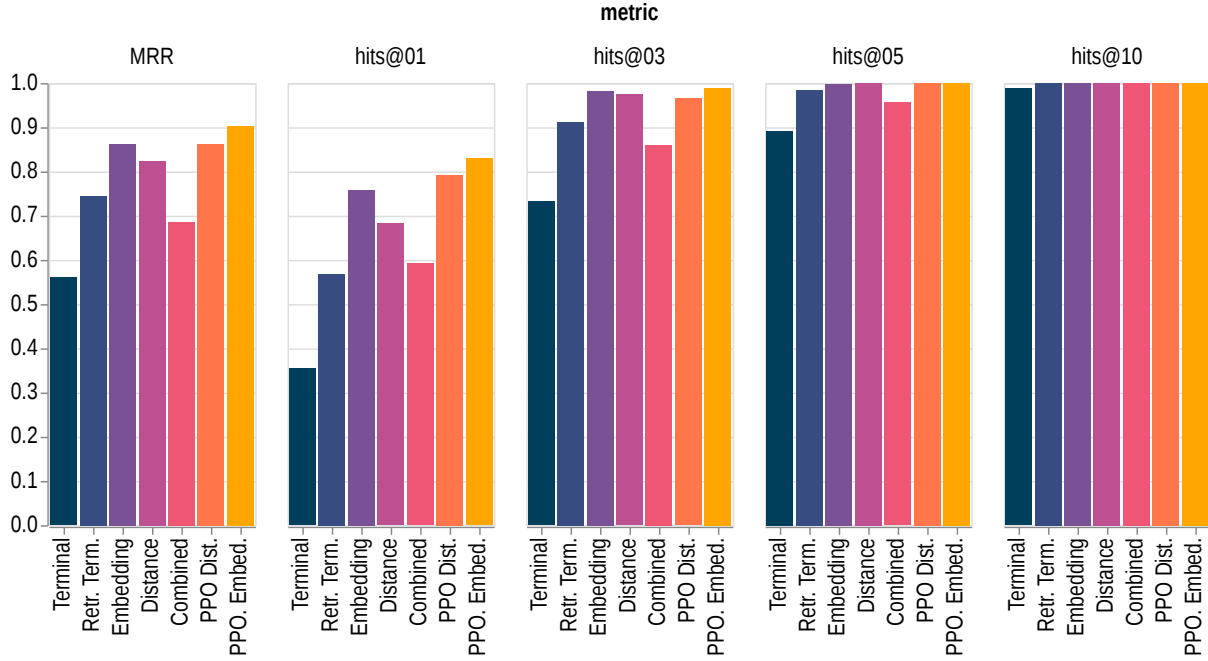
**metric**



**Figure 6.3:** comparison of metrics and techniques employed

cases in which $rank_e$ is above N.

| Algorithm | | Base | | | | PPO | | |
|---|---|---|---|---|---|---|---|---|
| Rewards → | **Retroprop** | | | **Simple** | | | | |
| Metrics ↓ | Terminal | Terminal | Embedding | Distance | Combined | Shaping | Distance | Embedding |
| hits@1 | 0.568 | 0.355 | 0.758 | 0.682 | 0.592 | 0.575 | 0.792 | 0.830 |
| hits@3 | 0.912 | 0.732 | 0.982 | 0.974 | 0.860 | 0.759 | 0.965 | 0.989 |
| hits@5 | 0.984 | 0.892 | 0.998 | 1.000 | 0.956 | 0.899 | 1.000 | 1.000 |
| hits@10 | 1.000 | 0.988 | 1.000 | 1.000 | 1.000 | 0.991 | 1.000 | 1.000 |
| MRR | 0.745 | 0.560 | **0.863** | **0.824** | 0.686 | 0.657 | **0.863** | **0.903** |

**Table 6.3:** Algorithms, reward and propagation techniques comparison.

All our experiments were conducted on a computer equipped with an Intel Core i9-9900K CPU, 64GB of DDR4 RAM and an Nvidia RTX 3080-Ti GPU.

### 6.4.3   Results and discussion

Our results are separated into two categories depending on the size of the datasets they were obtained from. UMLS and COUNTRIES are classified as small datasets, they provide useful information used for experimentation on larger datasets such as FreeBase, WordNet or NELL which are costlier to train and evaluate agents on.

**Small datasets**

Figure 6.3 shows the results of applying the different strategies mentioned to the agents trained in the COUNTRIES dataset. We can see the improvement from leveraging embeddings and distance rewards over relying on backpropagations of rewards as a measure for good performance in the episode. It is also noteworthy that

the combination of the rewards results in a worse performance than using them separately as it confuses the agent into navigating conflicting paths based on the weight given to each of the reward components.

As shown in Table 6.3, the different combination of strategies resulted in a notably higher MRR in four of them (highlighted in black). Therefore, we focused on further studying these approaches by using larger and more complex datasets: NELL, FreeBase and WordNet, and performed a number of experiments on them.

For the UMLS dataset, we trained two agents for a low epoch count to show their ability for fast convergence. Specifically, we trained a distance-based and an embedding-based reward agent with the PPO algorithm relying on the Actor-Critic implementation for them.

| Method | Mean | Std dev. |
|---|---|---|
| Terminal | 2.81 | $\pm$ 2.14 |
| Retro. term. | 1.76 | $\pm$ 1.34 |
| Embedding | 1.45 | $\pm$ 1.03 |
| Distance | 1.77 | $\pm$ 1.22 |
| Combined | 2.14 | $\pm$ 1.40 |
| PPO dist. | 1.71 | $\pm$ 0.83 |
| PPO emb. | 1.41 | $\pm$ 0.81 |

**Table 6.4:** Mean of ranked path with answer entity.

Table 6.4 shows the mean values of the ranking given by the agents to the inferred paths up to a maximum of 5. The majority of the generated paths are close to the top (value of 1) on average, and not even the base algorithm performs under 3. It is also noticeable how the standard deviation for both Actor-Critic methods is significantly lower than for the other models; as expected, the critic allows the method to converge faster, so for agents that train in the same conditions they offer a safer alternative that would generally perform better at any task.

| | Hits@1 | Hits@3 | Hits@5 | Hits@10 | MRR |
|---|---|---|---|---|---|
| Embedding | 0.148 | 0.424 | 0.620 | 0.852 | 0.339 |
| Distance | 0.116 | 0.396 | 0.608 | 0.836 | 0.312 |
| Improvement(%) | 27.586 | 6.604 | 1.935 | 1.878 | 7.965 |

**Table 6.5:** UMLS dataset metrics on a short training cycle.

Table 6.5 displays the results from the two PPO agents trained for the UMLS dataset with embedding and distance rewards respectively. This training was conducted for a small epoch count (52,160 iterations, 17,386 epochs) to test the convergence rate of the agents, which for a highly connected dataset such as UMLS indicates that PPO can obtain satisfactory results even in disadvantageous conditions. If we compare these

results to those in Table 6.2, in which agents were trained up to gradient convergence, we find that the results are similar. However, it took merely $1/50th$ of the time to obtain them, since REINFORCE with terminal reward takes orders of magnitude more epochs to train.

**Large datasets**

We trained several agents to compare algorithm and reward implementation combinations using the FreeBase and NELL datasets, and compared generalist agent performance versus relation-specific agents using the WordNet [55] dataset. These results show that our proposal is able to deal with web-scale datasets.

When using FreeBase, we trained four agents per algorithm and reward combinations with "film/film/genre" as the selected relation for a total of 159,896 episodes, or 53,299 epochs.

We trained a total of 12 NELL agents, four for each of the following relations: "thing_has_color", "is_taller" and "music_artist_genre" trained for 57,500, 64750 and 115,950 episodes respectively, as shown in column #ep from Tables 6.6 and 6.7.

In these tables, #rel denotes the frequency of appearance of these relations for each dataset, and #laps indicates the number of times the dataset was iterated during training.

| Alg. | Rew. | Hits@1 | Hits@3 | Hits@5 | Hits@10 | MRR | # rel. | # laps | # ep. |
|------|------|--------|--------|--------|---------|-----|--------|--------|-------|
| Base | Dist | 0.133 | 0.252 | 0.315 | 0.38 | 0.2567 | | | |
| Base | Emb | 0.167 | 0.282 | 0.348 | 0.405 | 0.2672 | 7268 | 22 | 159896 |
| PPO | Dist | 0.232 | 0.376 | 0.423 | 0.516 | 0.3829 | | | |
| **PPO** | **Emb** | **0.25** | **0.395** | **0.447** | **0.552** | **0.4082** | | | |

**Table 6.6:** Algorithm and embedding comparison for FreeBase dataset and "film genre" relation

Table 6.6 shows a comparatively low accuracy in juxtaposition to the small datasets for the "film/film/genre" relation, however the difference in accuracy for the PPO Agents is notable: when compared to the Baseline algorithm and distance reward hits@1 increases by 87.97%, and the Mean Reciprocal Rank by 59.02%.

Additionally, the use of embeddings with the PPO algorithm leads to an improvement of 7.76% and 6.2% for hits@1 and MRR respectively, which also shows the increase in accuracy for embedding-based rewards in highly connective datasets such as FreeBase.

In this case, the application of PPO shows a remarkably larger improvement in the agents results which is to be expected given the results in the former experiments. Embedding-based rewards also show improvement against the baseline of Euclidean distance, which confirms our intuition.
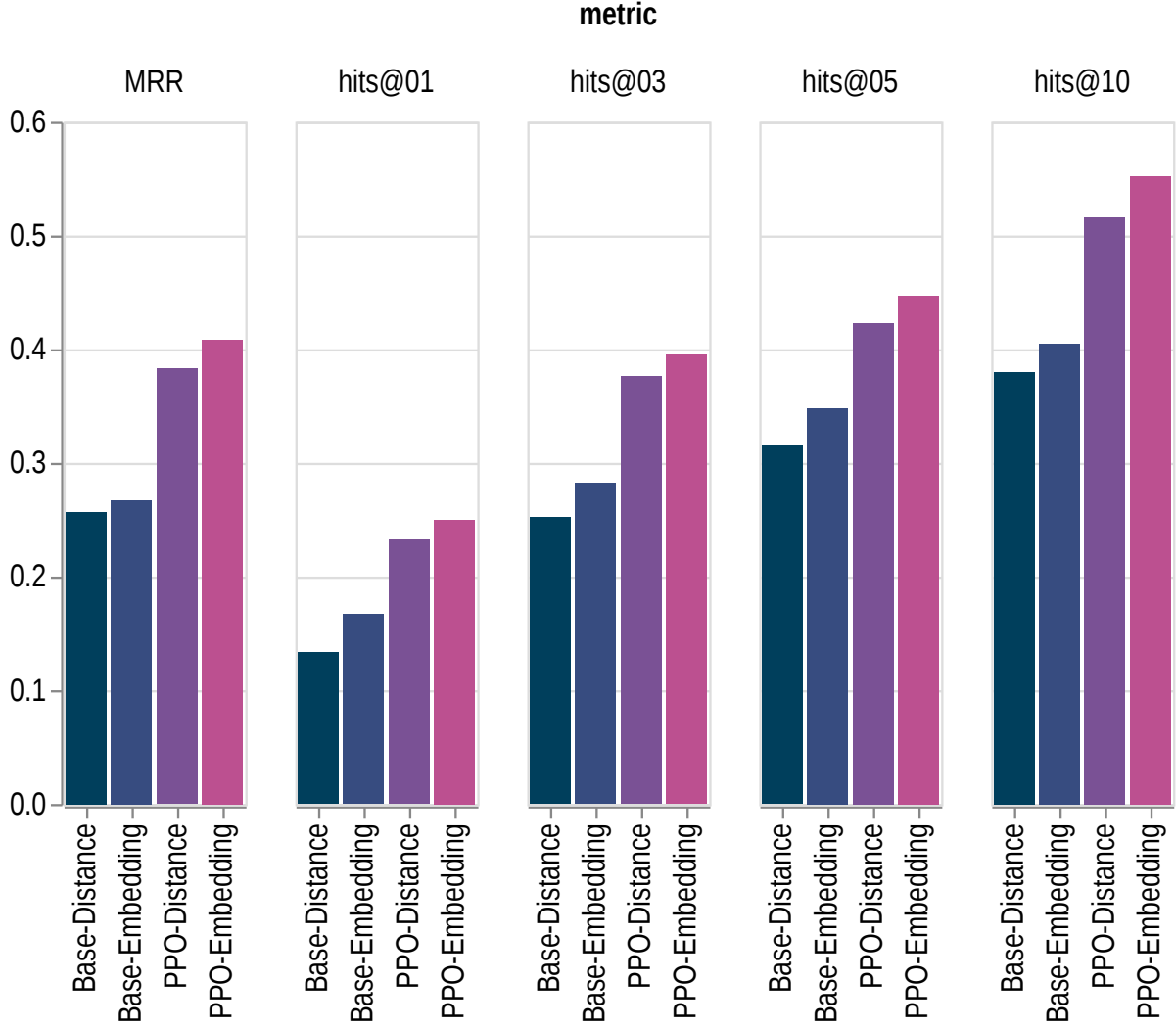
**metric**



**Figure 6.4:** Algorithm and reward comparison for FreeBase "film genre" relation

| Rel. | Alg. | Rew. | Hits@1 | Hits@3 | Hits@5 | Hits@10 | MRR | # rel. | # laps | # ep. |
|---|---|---|---|---|---|---|---|---|---|---|
| thing has color | Base | Dist | 0.445 | 0.804 | 0.908 | 0.949 | 0.6908 | 230 | 250 | 57500 |
| | Base | Emb | 0.5879 | 0.896 | 0.963 | 0.986 | 0.7710 | | | |
| | PPO | Dist | 0.551 | 0.869 | 0.947 | 0.983 | 0.7335 | | | |
| | PPO | Emb | 0.634 | 0.953 | 0.996 | 0.999 | 0.7889 | | | |
| is taller | Base | Dist | 0.410 | 0.770 | 0.873 | 0.915 | 0.6909 | 259 | 250 | 64750 |
| | Base | Emb | 0.585 | 0.893 | 0.960 | 0.983 | 0.7690 | | | |
| | PPO | Dist | 0.548 | 0.866 | 0.944 | 0.980 | 0.7313 | | | |
| | PPO | Emb | 0.630 | 0.950 | 0.993 | 1.000 | 0.7868 | | | |
| music artist genre | Base | Dist | 0.409 | 0.769 | 0.857 | 0.898 | 0.6496 | 773 | 150 | 115950 |
| | Base | Emb | 0.583 | 0.892 | 0.944 | 0.967 | 0.7599 | | | |
| | PPO | Dist | 0.547 | 0.865 | 0.928 | 0.964 | 0.7460 | | | |
| | PPO | Emb | 0.629 | 0.949 | 0.976 | 0.984 | 0.7755 | | | |

**Table 6.7:** NELL metrics for several relations.

NELL dataset agents were trained in relations with fewer instances than their FreeBase counterparts: 230, 259, and 773 instances of "thing_has_color", "is_taller" and "music_artist_genre" respectively against 7268 instances of "film/film/genre" in

FreeBase, meaning the agents are effectively trained faster due to converging earlier.

Even if FreeBase agents are trained for more episodes than NELL agents (159,896 in FreeBase versus 57,500, 64,750, and 115,950 in NELL), NELL agents better assimilated the structure of the dataset. This happens since the exploration triples $(e_0, r_p, e_1)$, which are the ones that contain the relations they are tasked to predict $(r_p)$, are used ten times as often for the same amount of episodes, as denoted by the #laps column in Tables 6.6 and 6.7.



(a) NELL - is_taller

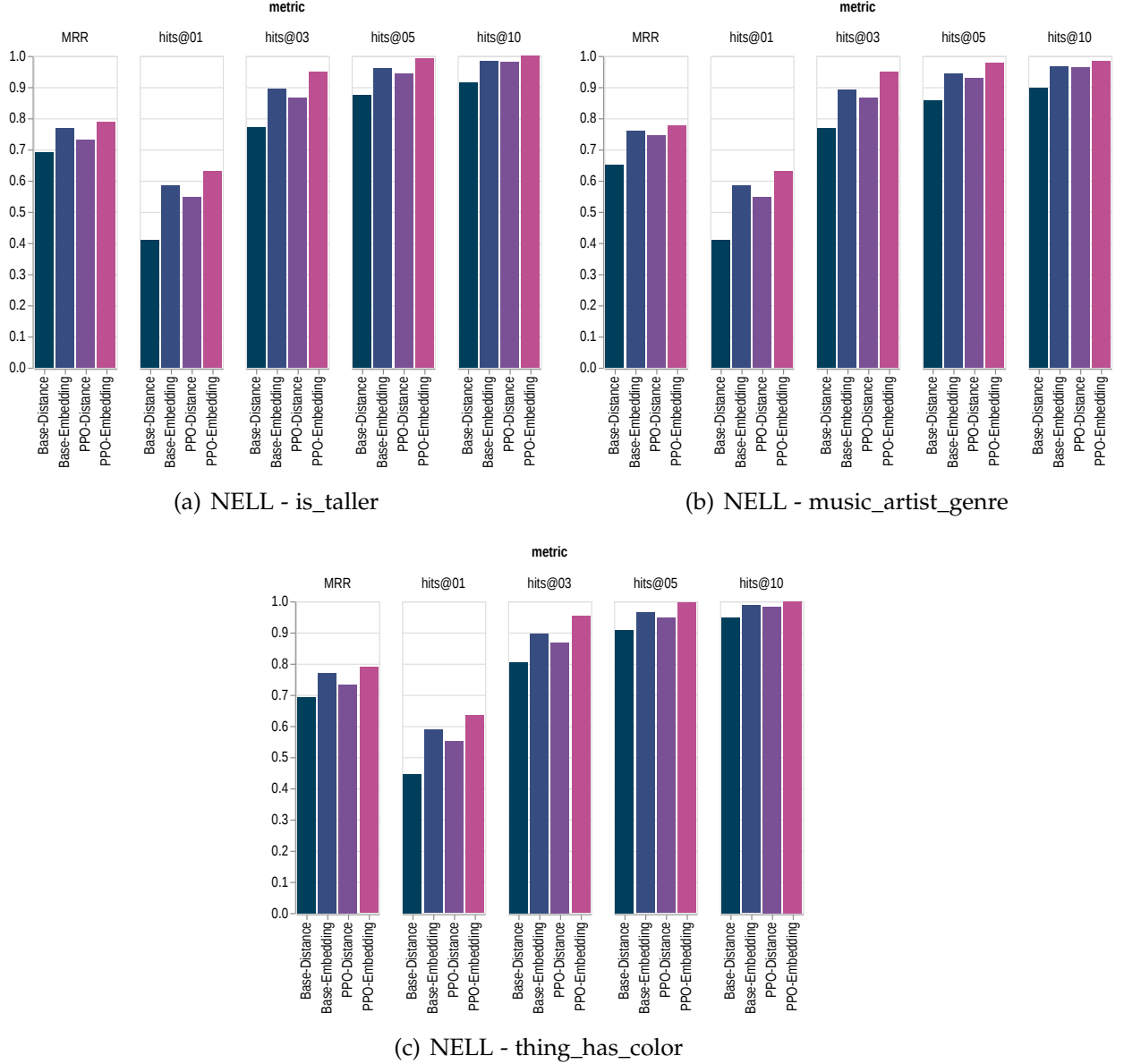(b) NELL - music_artist_genre

(c) NELL - thing_has_color

**Figure 6.5:** Metrics comparison for NELL dataset relations.

In this context, we make the following observations regarding improvements in performance for the "PPO + Embedding" (Best performance) versus the "Base + Distance" (Worst Performance) approach:

- In **"thing_has_color"** we see an improvement of 29.81% for the hits@1 and 16.23%

for MRR

- In **"is_taller"**, there is an improvement of 34.92% for the hits@1 and 13.02% for MRR
- In **"music_artist_genre"** the improvement is of 34.98% for hits@1 and 13.04% for MRR

The following listing shows the embedding versus distance reward improvement for the PPO algorithm. By leaving the algorithm unchanged, we focus only on the influence that the reward structure made on the experiments.

- For **"thing_has_color"** the improvement is 13.09% for hits@1 and 7.02% for MRR
- In **"is_taller"** we observe an increase of 12.19% for hits@1 and 6.77% for MRR
- In **"music_artist_genre"** the change was 16.23% for hits@1 and 3.8% for MRR

We can observe a general increase of 33.24 % for hits@1 and 13.82% for MRR in Base + Dist. versus PPO + Embeddings; in this case all agents reach convergence so the improvement is less noticeable than in the FreeBase scenario. The improvement achieved by selecting embedding-based rewards as opposed to distance-based ones (within PPO) is 14.11% for hits@1 and 5.86% for MRR, which shows that embedding-based rewards help increase the agents accuracy and reduces the variance significantly versus Distance based rewards.

| Rel. Name | Hits@1 | Hits@3 | Hits@5 | Hits@10 | MRR | # rel. | # laps | # ep. |
|-----------|--------|--------|--------|---------|-----|--------|--------|-------|
| similar to | 0.830 | 0.995 | 1.000 | 1.000 | 0.909 | 80 | 500 | 40000 |
| verb group | 0.609 | 0.934 | 0.989 | 0.999 | 0.771 | 1138 | 150 | 170700 |
| also see | 0.646 | 0.715 | 0.885 | 0.935 | 0.745 | 1299 | 200 | 259800 |
| deriv. rel.* | 0.585 | 0.651 | 0.746 | 0.933 | 0.6375 | 29715 | 100 | 2971500 |
| GENERIC | 0.405 | 0.484 | 0.523 | 0.660 | 0.443 | 80798 | 75 | 6059850 |

**Table 6.8:** Experimentation results with WordNet dataset
*derivationally related from

We used the WordNet dataset to test the accuracy of a general model against a relation-specific model, as well as to offer a comparison against the previously trained agents. We focused solely on embedding-based rewards and PPO algorithm model in this case, since they have achieved the highest performance in the previous experiments.

Table 6.8 shows that even when all agents have reached convergence, their accuracy drops depending on the frequency of appearance of the relations in the graph. We can further visualize this phenomenon by focusing on the inverse co-relation shown by the regression lines in Figure 6.6, where we see a downward trend on MRR 6.6(b) when the number of appearances of a relation is ascending in Figure 6.6(a).

This tells us that whenever the number of relations increases, the MRR (which shows the stability of the agent) drops, meaning it is harder for the agent to determine a good result from the pool of possibilities. However, we see that the increase is
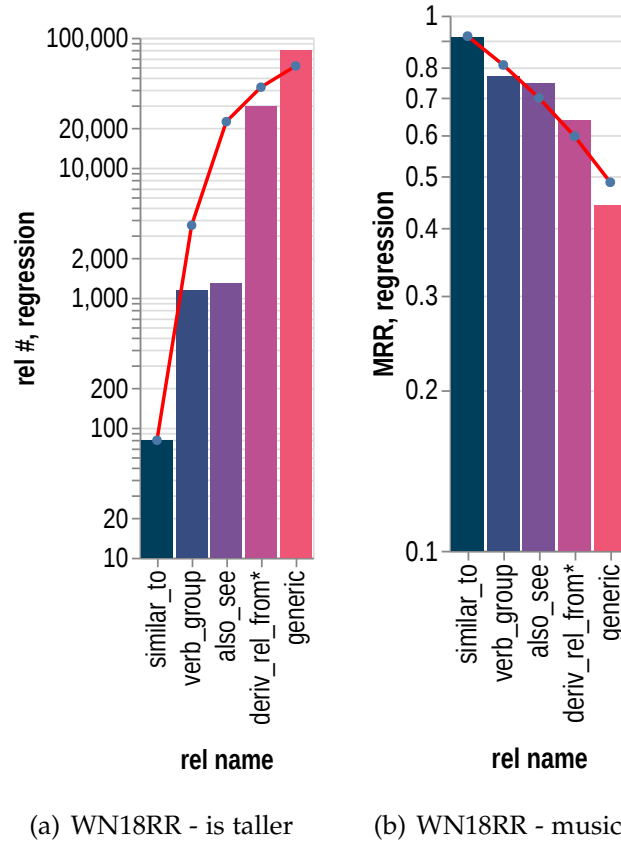
(a) WN18RR - is taller    (b) WN18RR - music

**Figure 6.6:** inverse relation for number of relations and MRR metrics for WordNet dataset.

*derivationally related from

logarithmic in nature, so it is reasonable to assume that it would still perform satisfactorily in adverse conditions.

## 6.5   Limitations

The performance of RL proposals heavily depends on the measurements of action performance (reward function) and the application of different policies that determine how the reward is ultimately assigned to the actions. However, existing proposals have merely scratched the surface when it comes to these aspects, applying unanimously terminal-based reward functions and the backpropagation-focused REINFORCE algorithm.

Our novel alternative consists of a new set of reward-functions and the application of RL algorithms whose potential remained unexplored. Our reward functions seek to use graph-specific information that is available before reaching the end of an episode: the distance to the answer node, and the semantic similarity to it computed from node embeddings. The implemented technique makes use of Proximal Policy Optimization and the Actor-Critic paradigm, resulting in faster training.

Both the new reward functions and policies have resulted in improvements over the state-of-the-art standard practices, particularly when using embedding-based reward functions on five widely used datasets. These results should motivate the development and evaluation of more variants of these aspects, since there is margin for improvement. Therefore, two trends of future work could be developed: 1) evaluating existing context-independent RL techniques, which are often already implemented by existing libraries but mainly remain untested in this context; 2) implementing new reward functions that make use of additional information in the graph, e.g. node attributes, which provide additional rich data.

## 6.6 Summary

In this chapter we have introduced SpaceRL, our proposal to complete knowledge graphs while overcoming several problems present in literature, focusing on a novel reward function and modern Reinforcement Learning algorithms. SpaceRL offers an end-to-end proposal to complete KGs while leveraging Reinforcement Learning to train intelligent agents capable of responding to queries by driving throught the graph and forming reasoned paths of information that respond to these queries, forming new explainable knowledge in the process. SpaceRL was tested against state-of-the-art datasets and performed significantly better with embedding based rewards and PPO RL-algorithms than other state-of-the-art approaches using well known an widely accepted ranking metrics.

# SpaceRL framework

*"The introduction of many minds into many fields of learning along a broad spectrum keeps alive questions about the accessibility, if not the unity, of knowledge."*

— Edward H. Levi

The importance of improving upon the works of others and not just mere replication in scientific research is one of the main pillars of the field. Multiple proposals focus only on publishing their results and not on advancing or even demonstrating the claims they presented in their publications as true, For this reason, we present our works as a framework, usable in multiple levels of expertise and meant for expansion. This chapter introduces our framework for KG reasoning and is structured as follows: Section 7.1 Introduces the framework to the reader, Section 7.2 describes the different components that constitute the framework, Section 7.3 offers an example of use for the framework, Section 7.4describes the possible future for the software, and Section 7.5 closes the proposal.

# 7.1  Introduction

Several software tools have been proposed to deal with the incompleteness in KGs. Most of them do so by leveraging the information in the graph to infer new triples that represent missing information [10]. The approaches in this area can be classified into four main categories, namely:

- Rule-based reasoning [7, 25, 40] focuses on finding correlations between existing entities in a KG, which when combined with logical operators may infer an existing albeit non-explicit relation between them. This way, the nodes that represent those entities in the graph can be linked, resulting in new triples for the KG. Despite their simplicity, rule-based techniques usually display lower performance than other approaches, since they ignore essential features related to the KG structure.
- Embedding models [15] transform KG elements into numerical vectors in an N-dimensional space. Embedding-based algorithms then rank the candidate tail entities for a given query $(s, r, ?)$, based on their distance to $s$ in the vector space, and retain the top $k$ candidates. The embeddings generated in this type of algorithms can additionally be used in combination with other techniques (such as Reinforcement Learning), since they provide a compact and informative representation of KG entities and relations. The main drawback of these techniques is that adding new triples to the KG usually entails having to re-compute the embeddings, which is usually a costly procedure.
- Relation path reasoning [8, 27, 53] focuses on finding paths that indirectly relate two disconnected nodes in a KG. Path-based algorithms build these paths by traversing the graph, and then discern which of those paths actually represent a specific type of relation between the node entities, which is then introduced in the KG as a new triple. The main benefits of these techniques are that they leverage the structure of the KG, resulting in a better performance, and that every path that results in a new triple provides additional explainability for the triple. The main limitation of this type of proposals is that traversing a complete and densely connected web-scale KG can be unfeasible; actually, some of this proposals use a random walk approach, which improves the scalability of the techniques, to the expense of ignoring promising paths in some cases.
- Reinforcement Learning (RL) [107] path finding enables multi-hop reasoning by using agents to find a path from a source entity to a tail entity that answers a given query. The policy-based RL agent learns to traverse the KG travelling from one entity to another adjacent one, and selecting in each step which link to follow, such that this decision maximizes the total episode reward. This can be described as a Markov decision process (MDP) which guarantees stochasticity, meaning that the process is non-deterministic.

From the previous analysis, we can conclude that Reinforcement Learning path finding is the most promising approach, since it leverages the KG structure, provides the same type of explainability as relation path reasoning proposals, while overcoming the scalability problem, and minimizes the risk of neglecting the most promising paths. Furthermore, it can be combined with embedding models to optimize the decision making in each step. Those reasons motivated us to make SpaceRL a RL-based tool for KG reasoning and completion.

There are some previous proposals in this field, such as DeepPath [107],MINERVA [16],Reward Shaping [45], PGPR [105], or DAPath [95]. However, these approaches require the embedding models to be computed beforehand, which hinders their performance. Also, they are restricted to using classical RL algorithms, and overlook the application of more modern RL algorithm such as Proximal Policy Optimization (PPO) [73] or Soft Actor Critic (SAC) [29]. Finally, most of these proposals are not distributed as usable tools intended for final users. Even if they make their implementation publicly available, their code is generally intended for the sake of reproducibility of their experimental results, and they often lack any degree of customization or flexibility, meaning they usually can only work on a number of predefined datasets as input.

SpaceRL combines the benefits from RL pathfinding with the power of representational embeddings to infer fairly long and explainable paths, useful for KG-based applications, and it can do so with on-the-fly embedding generation, which means that the KG embeddings are not a required input to the system.

Our tool is highly configurable, allowing for reward calculation to be modified with a combination of several options, customizing the policy intermediate activation function and regularization, using the more classical approach of the REINFORCE [89]. algorithm instead of PPO if required, computing the reward in one of several ways, or selecting the max depth of paths to explore, among other options. Also, SpaceRL allows the user to apply state-of-the-art RL algorithms out of the box, namely Proximal Policy Optimization (PPO)[73] combined with Soft Actor Critic (SAC) [29], which improve performance and help avoid reward plateaus while training. To the best of our knowledge, this is the first tool to provide such a wide variety of options.

Finally, SpaceRL, aims to provide a versatile tool intended for users with different levels of expertise, from novice to experts. It allows comprehensive and flexible customization for advanced users, who may prefer to install SpaceRL as a server for their local usage, or to become a service provider for third parties. On the other hand, it also offers a simple MLaaS interface, intended for a more untrained end user. Machine Learning as a Service (MLaaS) has gained traction in recent years [68], since it adds layers of abstraction that create a black box simplified interface for a non-expert final user. SpaceRL offers RL model generation and usage as a service capabilities, either

locally through its GUI or as a deployable REST API for third party consumption. Therefore, it is, to the best of our knowledge, the first turnkey tool to provide such RL KG completion and reasoning functionalities.

## 7.2   Software description

SpaceRL is an end-to-end Knowledge Graph completion tool, written entirely in Python and accessible to multiple user groups with different levels of expertise. SpaceRL provides an easy to use GUI tool for novices, an API for internal network or third party usage, and direct access to the low level application for advanced users or potential contributors. A diagram illustrating its internal code structure is depicted in Figure 7.1.

SpaceRL was designed to provide a wide variety of functionalities related to KG completion, including: embedding generation, path reasoning model training and testing, integration of pre-trained models, performance metrics calculation and reports, mock KG generation, and cache file generation for distance rewards, among others. Our software includes as well a graphical visualization tool to allow the user inspect the resulting paths inferred from a KG, providing additional information about the reasoning process.

To perform these tasks, SpaceRL requires only a KG as input, which is used to generate different embedding representations of the graph nodes and edges. In its current version, SpaceRL provides support for four classical embedding models in the literature, namely, ComplEx [97], DistMult [110], TransE [6], and TransR [46]. The effectiveness of these models has been confirmed by multiple authors; notwithstanding, further embedding models could be added to SpaceRL in the future. To implement these models, we relied on DGL-KE [114] a package which provides off the shelf embedding vector generation.

Our tool is publicly available at our GitHub page [1] and it is open to contribution. The application is divided into several subsystems which will be explored in the following subsections, and the interconnection between these subsystems can be seen in Diagram 7.2. They will be described from the perspective of an advanced user in order to provide a comprehensive view of the proposal.

### 7.2.1   Configuration

The configuration module of SpaceRL comprises two components: a key-value map which holds several global tuning parameters, and the `Experiment` and `Test` classes. The behaviour of the tool is defined by a list of class instances that describe the experimental and testing setups, respectively, and which can be found in the
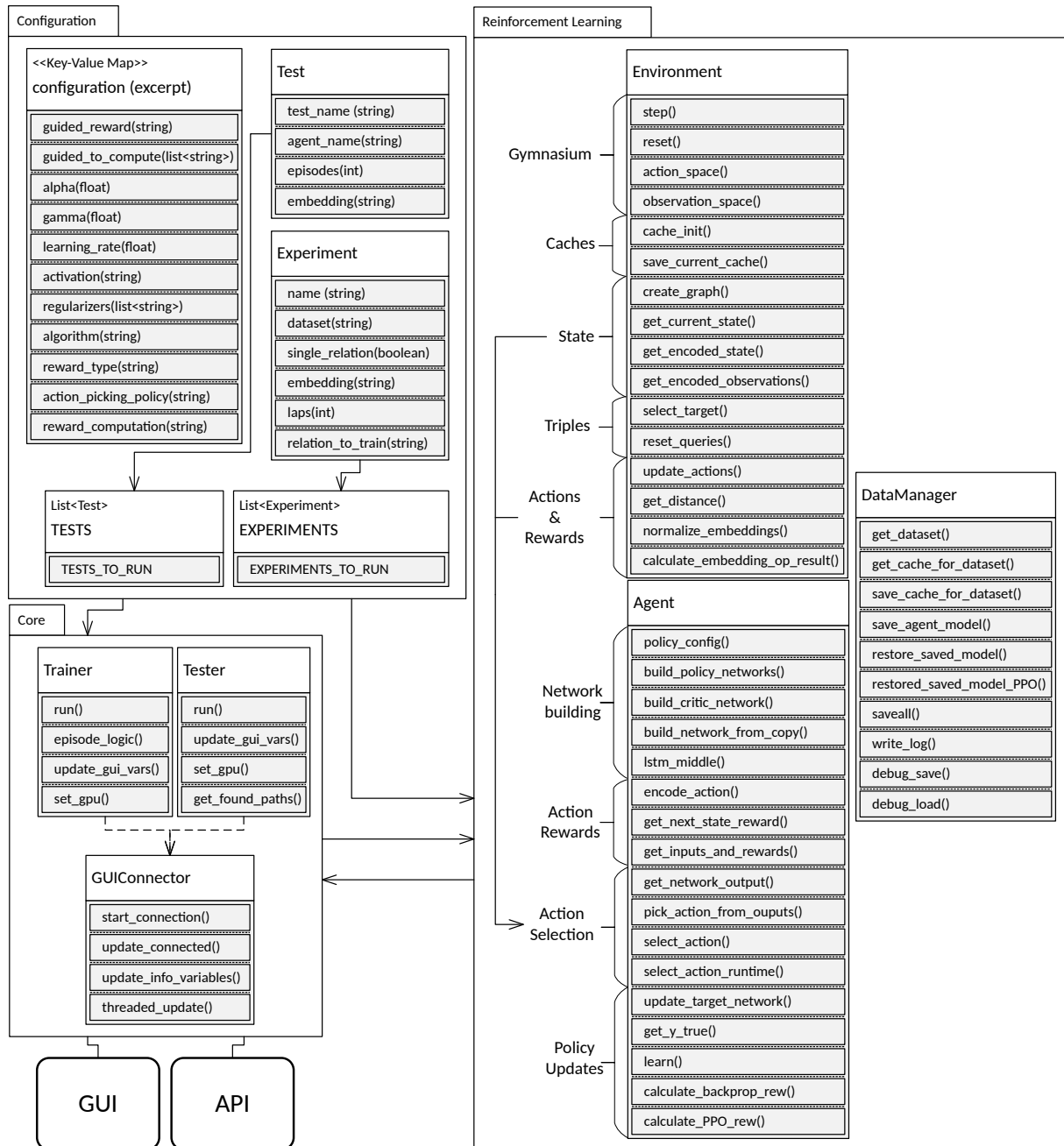
---

[1]https://github.com/DEAL-US/SpaceRL-KG

**Figure 7.1:** SpaceRL package diagram.

```python
EXPERIMENTS = [
    Experiment("wordnet_general", "WN18RR", ["TransE_l2","ComplEx"],200),
    Experiment("specific_rel_FB", "FB15K-237", ["TransE_l2"], 100, True, relation = "
    relation_name")
]

TESTS = [
    Test("wordnet_transE", "wordnet_general", ["TransE_l2"], 5000),
    Test("wordnet_complEx", "wordnet_general", ["ComplEx"], 10000),
    Test("FB_specific", "specific_rel_FB", ["TransE_l2"], 400),
]
```

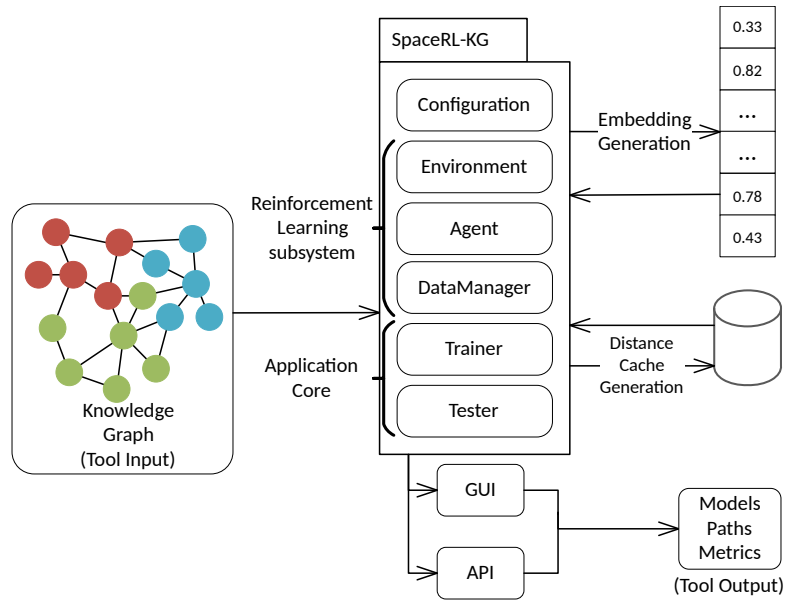**Listing 7.1:** Excerpt of config.py with experimental and testing configuration

**Figure 7.2:** SpaceRL work flow

`model/config.py` file (a sample excerpt of this file showing these lists can be found in Listing 7.1).

The most relevant global configuration parameters in the key-value map are the following:

- `activation` (`string`): Indicates what activation function to use in the agent intermediate layer. Available options are: `ReLu` [58], `leaky ReLu`[49], `PreLu` [30], `eLu` [12], or `tanh` [43].
- `alpha` (`float`): If using Proximal Policy Optimization (PPO), it defines the previous step learning rate.
- `action_picking_policy` (`string`): Indicates how actions are selected by the agent in every step of training. Available options are: `probability` and `max`.
- `episodes` (`int`): the number of episodes to train for.
- `gamma` (`float`): Decay rate of past observations, used only when `reward_type` is set to `retropropagation`.
- `guided_reward` (`boolean`): If set to `false`, a binary reward is used (reward value is either 1 or 0); otherwise, a step-based reward is used, as specified by the `guided_to_compute` parameter.
- `guided_to_compute` (`list<string>`): If `guided_reward` is set to `true`, the user can configure additional options:
  - `Terminal`: If the agent reaches the target node it overrides other rewards and sets it to 1.
  - `Shaping`: Performs embedding addition of node and relation embeddings to measure the distance from the current node to the target node in vector space.
  - `Distance`: Measures the distance from the agent to the target node.

- **Embedding**: Combines several vector embedding operations which determine how successful the agent chosen action was towards bringing it closer to the target node.
- `learning_rate` (`float`): Defines the policy neural network learning rate.
- `normalize_embeddings` (`boolean`): Normalizes the vector space after performing embedding regeneration.
- `path_length` (`int`): Specifies a maximum limit for the inferred paths length.
- `regenerate_embeddings` (`boolean`): Re-calculates specified embeddings vectors for the desired KG.
- `regularizers` (`list<string>`): Indicates during which training step L1 and L2 regularization should be applied. Available options are: `kernel`, `bias`, and `activity`.
- `reward_computation` (`string`): Indicates how to calculate the reward value passed on to the learn function to update the policy network based on the agents neural network output. Possible values are:
    - `max_percent`: Scales the agent output to [0, 1] where 1 represents the highest output for the step and 0 the lowest.
    - `one_hot_max`: Binary reward, 1 for the maximum reward in the episode, 0 otherwise.
    - `straight`: The output from the agent is passed on directly as the reward.
- `reward_type` (`string`): Indicates how the rewards are propagated to the agent in the learning phase. Available options are: `retropropagation` or `simple`.
- `use_episodes` (`boolean`): If set to `true`, the agent is trained for the number of episodes specified in the `episodes` parameter; otherwise, it relies on the `laps` value of the `Experiment` class.
- `use_LSTM` (`boolean`): If set to `true`, LSTM layers are added to the agent when generated.

Regarding the `Experiment` class, it is responsible for agent training, and it requires the following specific configuration parameters:

- `experiment_name(string)`: unique name for each agent.
- `dataset_name(string)`: name of the KG used for training.
- `embeddings(List<string>)`: embedding model used by the the agent, being the possible options: `TransE_l2`, `DistMult`, `ComplEx` and `TransR`.
- `laps`: number of laps that are taken by the agent around the KG in order to minimize randomness. Note that larger KGs may require more laps but this increases inference time linearly.
- `single_relation(boolean)`: if `true`, the agent trains for a single relation; otherwise, it does so for the entire collection of relations in the KG.
- `relation(string)`: the name of the relation to train for, if

`single_relation` is set to `true`.

The `Test` class is tasked with testing the performance of trained agents and generating reasoned paths from this operation, requiring the following parameters:

- `test_name(string)`: unique name for each test.
- `agent_name(string)`: name that identifies the agent that is going to be tested.
- `embeddings(List<string>)`: subset of the embeddings used by the agent during training, which will be used for testing.
- `episodes(int)`: number of tests to perform for the selected agent.

## 7.2.2  Reinforcement Learning

The RL subsystem takes a KG as an input, and it is responsible for the generation of the training environment, creating the agent that will train on the input KG with the specified configuration options, and managing the data generated during training and testing.

A Reinforcement Learning *environment* represents the context in which an agent will act and learn. The environment has a *state* that can be manipulated by a number of agent operations called *steps*. However, the agent has only a partial view on the environment in each step, which puts a limit on the actions it can take; this is referred to as the *action space* available to the agent in the current state. Figure 7.3 illustrates how the different Reinforcement Learning modules interact with one another.
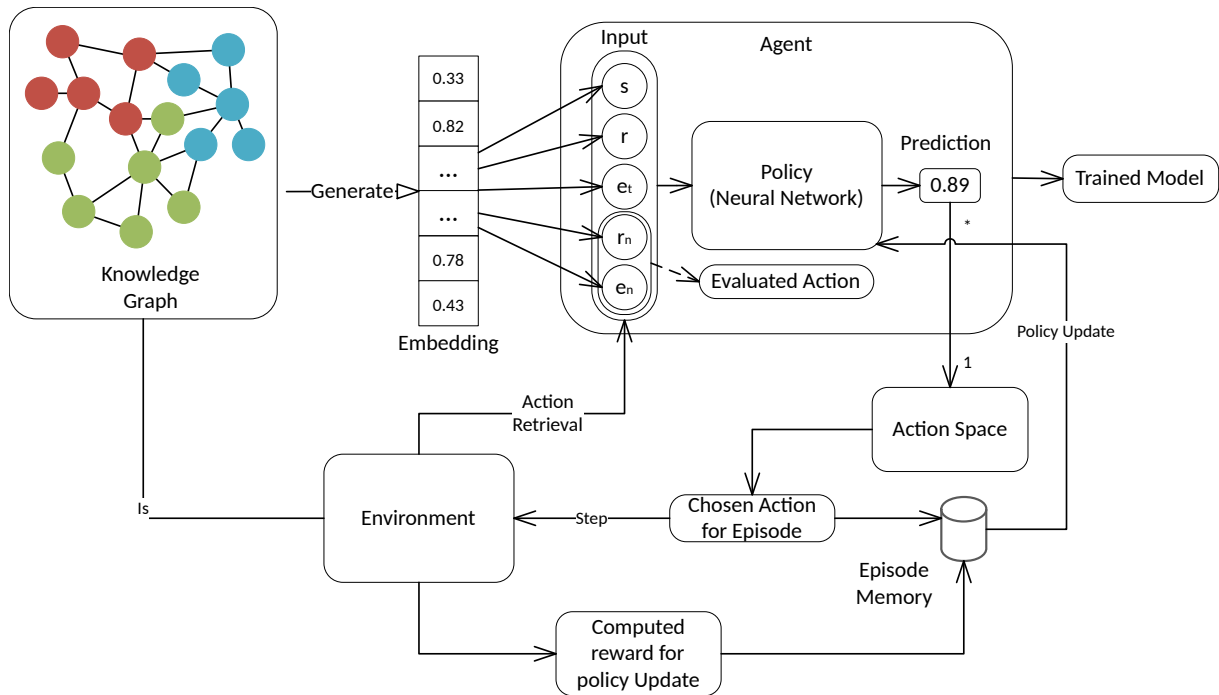


**Figure 7.3:** Reinforcement Learning subsystem work flow

The subsystem is comprised of a number of classes, namely: `Environment` (`model/environment.py`), `Agent` (`model/agent.py`) and `DataManager`

(`model/data/data_manager.py`). We will explore these classes in depth in the following subsections.

**Environment**

SpaceRL environment is built with navigating the KG in mind; to that effect, we consider that the environment is the entirety of the KG, the state is a particular node in the KG, and the action space is comprised of every relation that links that node with its adjacent nodes (which may include itself). To create an instance of the environment, a `DataManager` class instance is needed, as well as tuning some of the configuration parameters defined in Section 7.2.1: the KG triples, number of laps, and optionally, a specific relation to train for.

The `Environment` class was implemented following the OpenAI Gym [62] standard for reinforcement learning tools, recently transferred to the Farama Foundation and its new drop-in replacement Gymnasium[21]. Enforcing this standard entails the implementation of a number of functions, namely `reset`, `action_space`, `observation_space` and `step`, with the latter returning the environment state and a `done` flag to signal early stopping of the episode. Thus, it is guaranteed that the `Environment` class is responsible for state management and action generation. Our implementation also provides on-demand distance cache generation, consultation and reward generation, all of which are used by the agent to drive through the KG.

Once created, the `Environment` instance first invokes the `KnowledgeGraph` class, initializes the cache, and generates the KG embedding vectors if they are not present, as SpaceRL stores previously generated embedding models while also offering a module to pre-generate them if desired.

Every training episode begins with a query triple $(s, r, t)$, and the node that contains the head entity $s$ as the initial state of the episode. The selection of the episode initial triple is performed by the environment during the `reset` operation. Then, the link representing relation $r$ is then removed from that particular instance of training. Figure 7.4 offers a graphical representation of the environment, in which the current state is the initial query triple head entity. After the outgoing triple relation is removed, SpaceRL begins the training process.

For each training step, the environment encodes the initial state as a concatenation of embedding vectors, and calculates all possible actions starting from that state. Then, it relies on the `Agent` to select one of those actions in order to advance to the next state. The process is repeated for each state until the episode is complete. In that moment, the reward to be used for policy updates is also calculated based on the configuration parameters chosen.
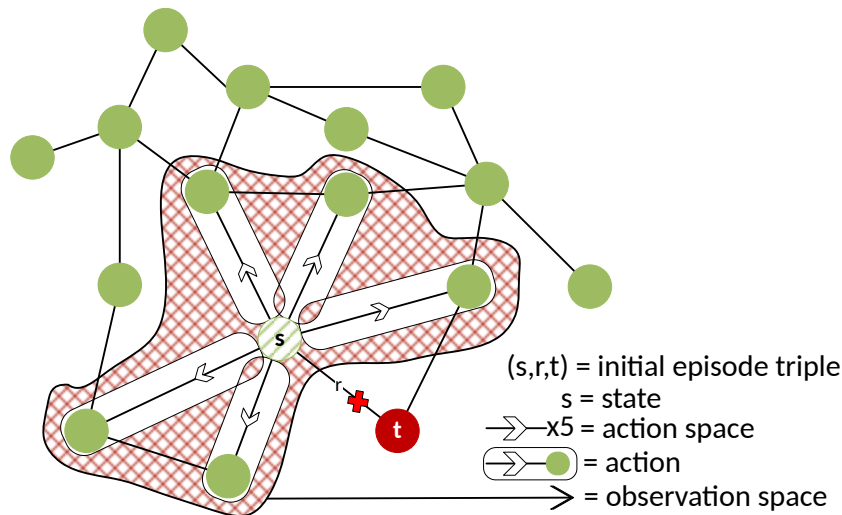
**Figure 7.4:** SpaceRL environment

**Agent**

The `Agent` class is one of SpaceRL most complex components, tasked with building the neural network according to a given specification, as well as selecting the actions on each step according to the `action_picking_policy` and `reward_computation` values in the configuration, memorizing them, storing the rewards given by the environment and advancing to the next environment step.

Creating an instance of the `Agent` class requires providing an instance of the `Environment` and `DataManager` classes, and some configuration parameters, such as the activation function for intermediate layers, the RL algorithm to use, the reward components to activate, and the hyperparameter values `alpha` and `gamma`.

Once instantiated, the agent initializes its memory and the neural network according to the given configuration parameters. It can do so following a PPO algorithm, which is implemented with an Actor-Critic strategy requiring multiple neural networks, or else by following a classic RL algorithm with a single network.

Subsequently, L1/L2 regularization is configured, the base NN layers are added and then complemented with intermediate LSTM layers [32] if requested. Finally, either ADAM [37] or RSMprop [28] optimizers are used. If PPO was the selected algorithm, a critic is built sharing its network architecture with the Actor network.

As was described in previous section, class `Agent` class intimately interacts with class `Environment` during the training episodes. For each action generated by the `Environment`, the `Agent` calculates the output of the neural network, which represents the score assigned by the agent to that action. Using the scores, the `Agent` class evaluates the former actions and selects one of them to proceed, which updates and steps the environment into the next state. A stopping condition is defined in order to command the agent to end the current training episode. Alternatively, the user may

specify a maximum number of steps as stopping condition.

Machine learning libraries Keras [36] and Tensorflow[92] are responsible for generating the neural network layered structure and then calculating the outputs on each step and allowing for simple storage and loading of trained models.

**Data Manager**

The `DataManager` class is responsible for data storage, modification, and organization during testing, training, and embedding generation processes.

This class handles KG processing by reading the input KG, expressed as a triple list file and the selected embeddings for training. It then builds two key-value maps, in which the keys are the entities and relations (respectively) for the selected KG and the values are the numerical vectors generated for each chosen embedding. It also handles cached distance rewards, periodically saving the distance cache file, updating it as the training goes on, and improving future agent training.

The `DataManager` class also handles persisting and loading agent models, logging information about the training, testing, debugging, and checking the integrity of the system in case an instance of training or testing ended abruptly leaving incoherent files behind.

In summary, the `DataManager` class provides the necessary information for other system components (specifically, for the `Environment` and `Actor` instances) and handles storage of that information in between episodes.

### 7.2.3 Core

The main subsystem of the application acts as the entry point to run a new training or testing process. Its main components are the trainer `model/trainer.py` and the tester, `model/tester.py`. In the following subsections, they are described in further detail.

**Trainer**

The `Trainer` class only requires the configuration key-value map mentioned in section 7.2.1 for its initialization. It first detects available GPUs in the system and configures itself to use them if allowed, then instantiates the `DataManager`, `Environment`, and `Agent` classes. Then, the `Trainer` awaits for the `run` method to be called, which triggers the training episodes. Each episode starts by obtaining one triple from the Environment, and performing the following actions:

- Reset the environment, by setting the state to the head node of the selected triple and removing the outgoing relation from the KG.
- Request the `Agent` class to select the most promising action to take, along with the reward for the selected action and the maximum reward for the episode.

- Advance one step in the environment and update it by taking the action chosen by the agent.
- Store the chosen action and its rewards in the agent memory.

If the environment activates the `done` flag, the episode ends, and the total reward per step is computed and passed onto the agent learning function. Finally, the neural network weights are updated based on the taken path and computed rewards.

**Tester**

The `Tester` class needs an already trained agent with which to perform the testing. The `Tester` receives the configuration map, the agent model which will be tested, and the embeddings and algorithm that were used during training, as well as the number of training episodes. Then, it performs the setting up operations, namely: configuring the GPU (if available), instantiating the `DataManager`, `Environment` and `Agent` classes, and preparing the models to operate in the environment by replacing the neural network model created for the Agent class with the ones received in the initialization of the `Tester` class.

During the testing process, the system iterates through the list of tests in the configuration file (`config.py`). For each of them, one agent model and `Tester` instance are loaded for each embedding specified in the test configuration. Then, the `Tester` executes the specified amount of testing episodes, as described in Algorithm 3.

For each episode, the Algorithm tries to infer new paths starting at the head node of the episode triple, and navigating until the maximum the length specified by the configuration parameter (`path_length`) has been reached. Once 10 paths have been found for the episode, they are ranked based on their scores. Note that we set the limit to 10 in order to be able to compute the Hits@10 metric value. Finally, the paths that actually reach the episode triple target entity are returned.

The evaluation metrics Hits@N and Mean Reciprocal Rank (MRR), are also computed and returned as output of the algorithm (note that we compute Hits@N for N=1, 3, 5, and 10). These are commonly used in the literature to evaluate the performance of ranking algorithms. Hits@N results represent the average performance of the agent over a number of testing episodes. It is calculated by obtaining the N first paths in each episode rank, and counting the number of positive episodes, i.e., episodes in which at least one of the N first paths actually reached the target entity. Then, the number of positive episodes is divided between the total number of testing episodes to get the overall Hits@N value for the agent. Function `add_to_hits_under_value` is responsible for this calculation.

MRR is obtained by averaging the rank position value for each path. This value is computed as

---

**Algorithm 3:** Testing algorithm

---

**Input:**
*env*: Environment // environment instance
*agent*: Agent // pre-trained agent instance
*config*: Map // configuration key-value map
*episodes*: Integer // total number of episodes
**Output:**
*paths*: List<path>// map of found paths
*hits*: Map<Int, Int >// map of raw number of hits per N
*mrr*: Float // calculated MRR for agent

1  *hits* ← {1 : 0, 3 : 0, 5 : 0, 10 : 0}
2  *paths* ← [ ]
3  *ranks* ← [ ]
4  **for** 1 *to episodes* **do**
5     *found_target* ← *False*
6     *local_paths* ← [ ]
7     **for** 1 *to* 10 **do**
8        *triple* ← *env.reset*()
9        *path* ←[*triple.head*()]
10       **for** 1 *to config.path_length* **do**
11          *action* ← *agent.select_action*()
12          *env.step*(*action*)
13          *path.add*(*action*)
14       *local_path.add*(*path*, *get_path_score*(*path*))
15    // rank paths according to their score
16    *local_paths* ← *sort_paths_by_score*(*local_paths*)
17    **for** *n*, *p* ← *enumerate*(*local_paths*) **do**
18       **if** *path_reached_target*(*p*) **then**
19          *hits* ← *add_to_hits_under_value*(*hits*, *n*)
20          *found_target* ← *True*
21          *ranks.add*(1/*n*)
22          *break*
23    // if target found, add top path to return list
24    **if** *found_target* **then**
25       *paths.add*(*p*)
26 *mrr* ← *ranks.sum*()/*episodes*
27 **return** *paths*, *hits*, *mrr*

---

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i}$$

where N is the total number of tests, and $rank_i$ is the rank of the top evaluated path that reached the target entity.

The textual representation of the returned paths is stored in a text file in order to be accessible by other processes.

### 7.2.4 Graphical User Interface

SpaceRL offers a GUI that provides the most important functionalities for an average user. The GUI implementation is based on Python default GUI manager Tkinter. An overview of the GUIs elements can be seen in Figure 7.5.
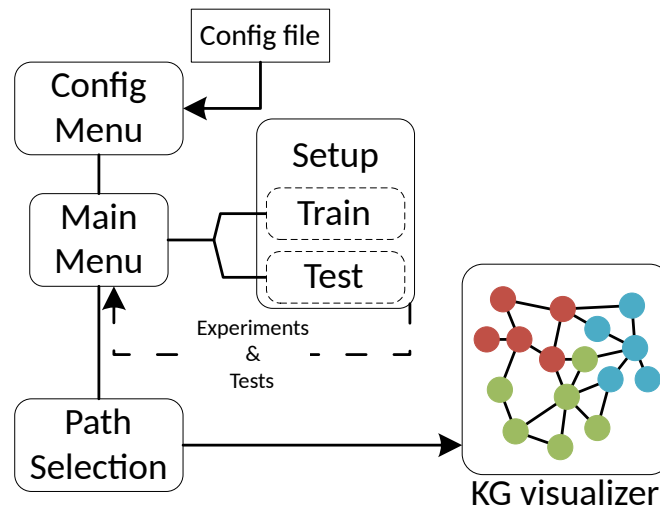


**Figure 7.5:** SpaceRL GUI structure

When the GUI is launched, the **main menu** is displayed, as depicted in Figure 7.6. The most relevant menu options are the following:
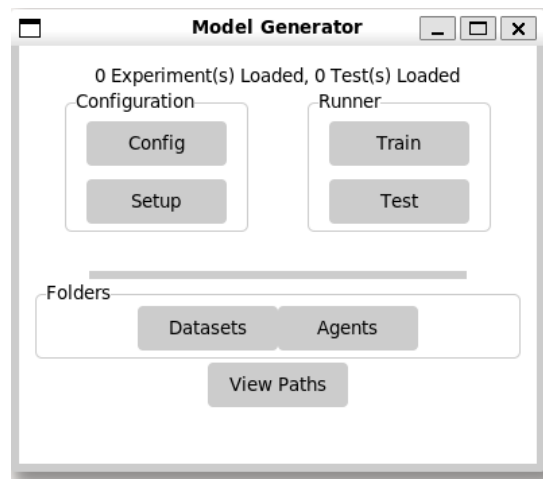


**Figure 7.6:** SpaceRL GUI: Main menu window

- The **Configuration** block, which allows the user to open the Config and Setup submenus.
- The **Runner** block, which presents the user the options to launch the training and testing processes, respectively.
- The **Folders** block, which allows the user to add and remove knowledge graphs. It also includes an option to import pre-trained agent models directly from h5 files, which are native to the Keras environment and preserve all model information.

- The **View Paths** option allows the user to generate visualized paths for the tested agents.

The **Config submenu** includes the global configuration parameters described in Section 7.2.1, organized into logical groups. The GUI provides some descriptive information about these parameters by means of tooltips, which help the user select the most convenient value for each of them.
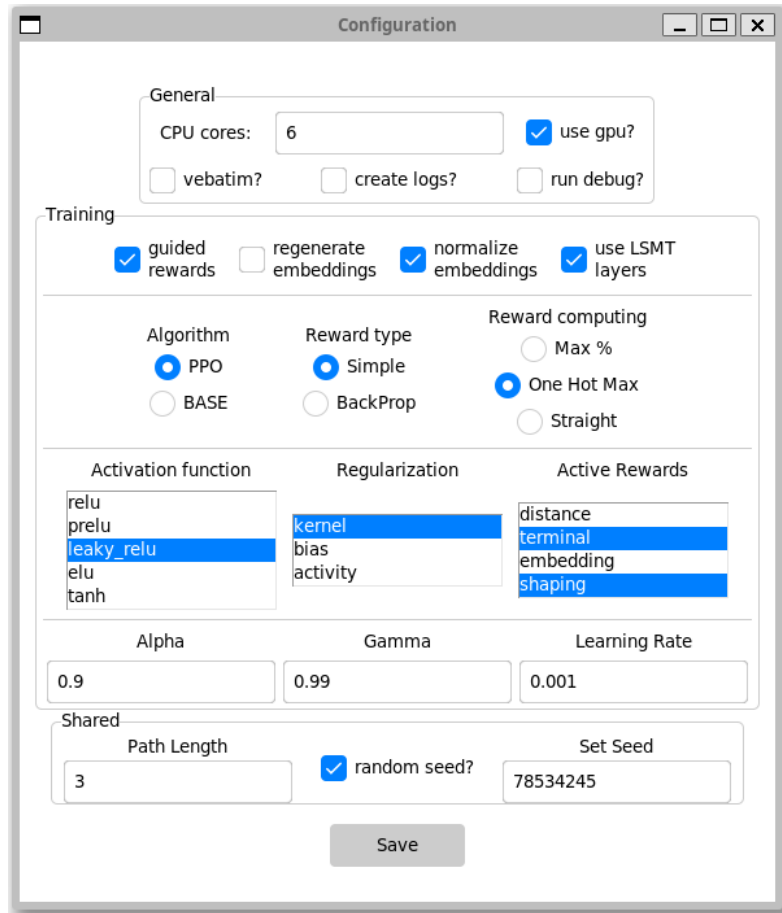


**Figure 7.7:** Configuration menu

The **Setup submenu** is divided into two tabs, **Train** (Figure 7.8(a)) and **Test** (Figure 7.8(b)), which include the specific parameters for the training and testing processes, respectively. The tool validates the user input, looking for common mistakes, such as specifying an already existing agent name, an unusually large number of laps, or a relation that does not exist in the given KG. Once the desired experiment or test has been configured, it can be added to the list using the **add** option at the bottom of each tab. The added elements are then listed at the right side of the corresponding tab, as seen in Figure 7.8.

After the training experiments and tests have been configured, they can be launched using the options available at the **Runner** block of the main menu (options **Train** and **Test**, respectively). The underlying modules are executed asynchronously by means of subprocesses, which enables SpaceRL to run them in the background, and provide
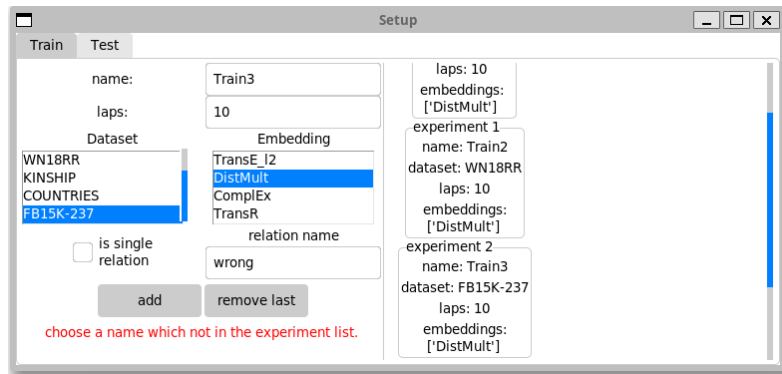
(a) Train submenu presenting 3 distinct experiments listed and an error being displayed
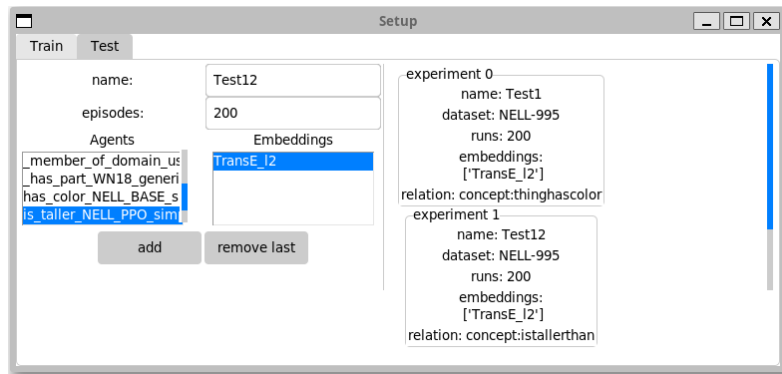


(b) Test submenu with several test listed and a suggested embedding for the selected test shown

**Figure 7.8:** SpaceRL GUI: Train and test submenus
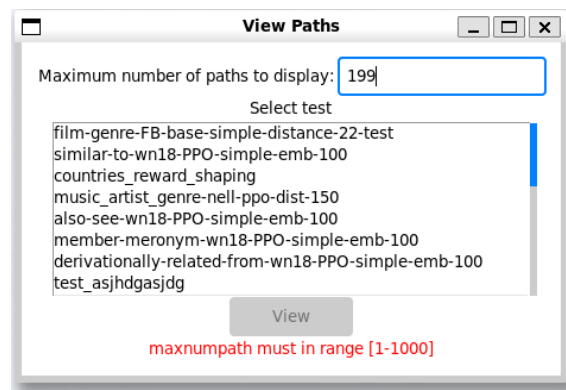
real-time progress update through cyclic polling.



**Figure 7.9:** SpaceRL GUI: Visualization menu

**Visualizer**

The visualization tool is accessed through the **View Paths** option in the main menu. First, a window is opened displaying the available test results, to allow the user select one of them. The user also needs to specify the number of paths to load from that test scenario, as seen in Figure 7.9 (in the current version, the number of paths is limited to

1000).

After clicking the **View** button, the visualization tool main window is opened, displaying one of the paths that led to the target triple and its scores in the selected test, as shown in Figure 7.10. The complete inferred path is displayed at the top of the window in plain text, as the sequence of entities and relations that compose the path. Below, a graph is depicted that illustrates the relevant nodes in the KG, with the available actions at each step.

Initially, the complete path is shown with only the name of the chosen relations and the score given to them by the agent (step 0). The user can then navigate between path nodes with the arrow keys, thus getting extended information about each subsequent step, namely: the node in which the agent was (highlighted in red), and the score given to each outgoing relation, which corresponds to a possible action that the agent could have taken. The selected actions are highlighted in a bold dark red line. Finally, the arrow buttons on the leftmost and rightmost sides of the screen allow the user to visualize the former and next path, respectively. Note that, in the current version, the paths are displayed to the user in the same order as the agent discovered them.

The implementation of our visualization tool is based on Pygame [66], which acts as a mediator between SpaceRL and the SDL kit [74] that offers low level access to keyboard, mouse, and graphics ardware. We also relied upon the NetworkX package [59], to compute the 2-dimensional positions of the graph nodes based on the Kamada-kawai distribution [35], and convert these positions into application pixels positions.
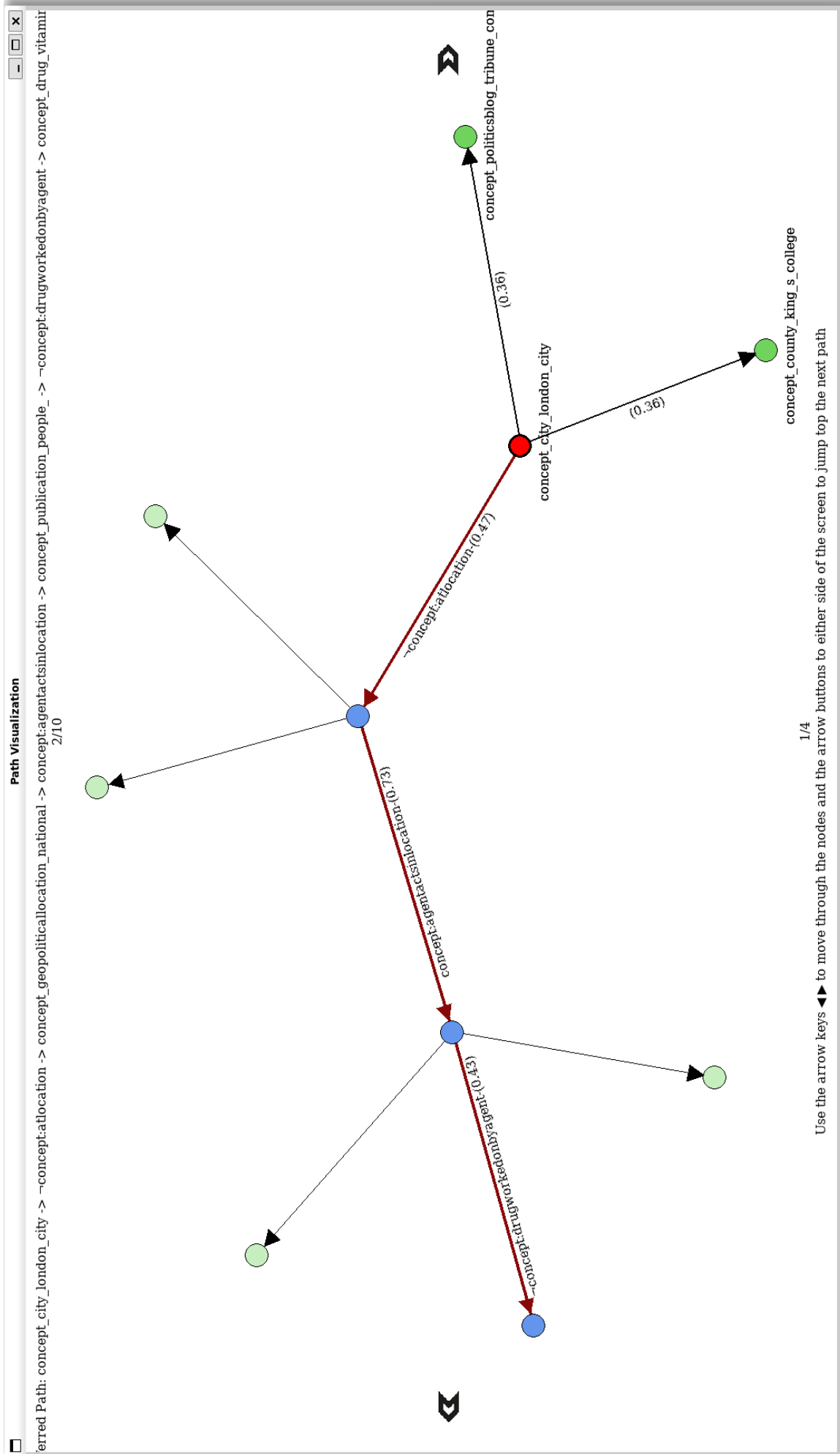
**Figure 7.10:** SpaceRL GUI: Visualization tool.

### 7.2.5 API

SpaceRL also provides an application programming interface that can be used by developers to implement their own applications on top of our functionalities. Our API service relies on FastAPI[22] to generate an OpenAPI [63] compliant application, and Uvicorn [99] as a backend webserver to host the app. The use of FastAPI makes it easier to deploy it to a production server or to exchange it for another backend if desired.
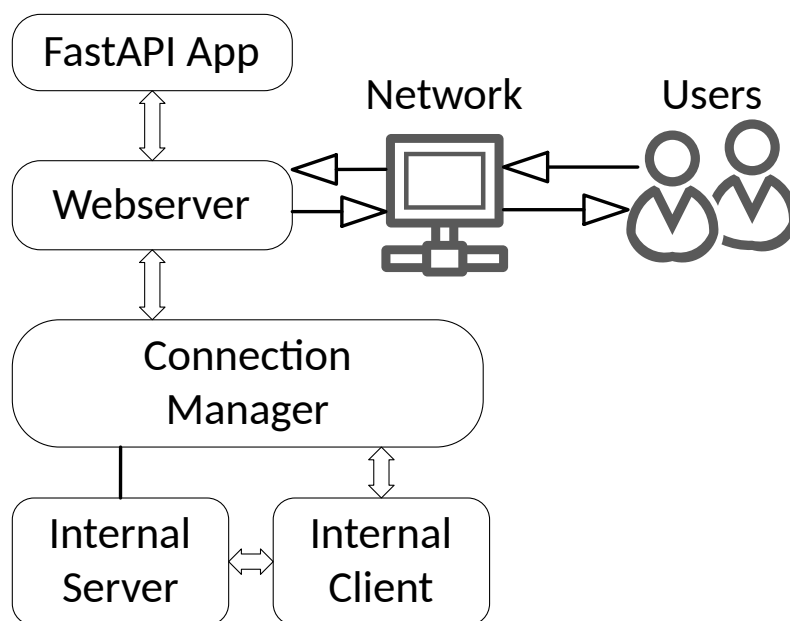


**Figure 7.11:** API structure of SpaceRL

As depicted in Figure 7.11, the FastAPI application handles user requests and our `ConnectionManager` class holds the internal client-server architecture, which is responsible for resource intensive operations while delivering a response back to the main application to show fast progress to the end user. The `ConnectionManager` class does not rely on intermediate packages to handle requests, which makes it faster and independent from interoperability limitations given by other general purpose software. This, however, adds complexity to the tool, since the implementation of the logic regarding encoding and decoding of requests was custom-made.

FastAPI requires the definition of custom classes to describe the responses returned to the user, as part of the OpenAPI specification. The classes we defined for SpaceRL are the following:

- **Triple** describes entity connection through a given relation.
- **Experiment** describes an experiment suite with a name, the KG to use, the embedding model to use, the number of laps to train, if it is focused on a single relation, and if so, which one.
- **Test** describes a test suite, including a name, the agent name to test, and the number of episodes.

- **EmbGen** is used to generate new embeddings model prior to any experimentation. It is not a necessary step but it makes the training process faster.
- **CacheGen** is used to acquire distance information about a particular KG. It streamlines the training process if distance rewards are used.



**Figure 7.12:** API endpoints

Figure 7.12 shows the web view of the API endpoints given by SwaggerUI [90].

SpaceRL internally distinguishes between two types of endpoints, *instant* and *process*. Both provide fast responses to the end user; however, *process* endpoints correspond to computationally intensive background tasks, which require a significant amount of resources. Therefore, the response sent to the user in those cases is merely a message to inform whether the process could be launched. If the resources needed to attend a *process* request are available, the `ConnectionManager` underlying client sends

a plain-text message to the internal server with the request in a particular format and the server answers back in the same way (cf. listings 7.2 and 7.3 for examples of these exchanges).

```
1 message: post;cache;{'datasets':['COUNTRIES'], 'depth': 3}
2 response: success;cache is being generated, please be patient
```

**Listing 7.2:** Internal server cache generation request and response

```
1 message: "post;embedding;{'dataset': 'NELL-995', 'models': [], 'use_gpu': True, '
    regenerate_existing': True, 'normalize': True, 'add_inverse_path': True, '
    fast_mode': False}"
2 response: "success; Embedding Calculation Launched"
```

**Listing 7.3:** Internal server embedding generation request and response

Next, we provide a description of the available endpoints in detail:

- /root (*instant*): the response is a welcome message as confirmation of a correct deployment, as seen in figure 7.13(b).
- /config (*instant*): it is used to retrieve (GET) or manipulate (PUT) the configuration key-value map. Only one parameter at a time can be modified, since validation is performed individually.
- /datasets (*instant*): it can be used to get the names of all KGs currently stored (GET), delete existing ones by name (DELETE), or create a new one (POST), by providing a number of triples in the request body.
- /cache: it allows retrieving the name of the KGs that have an associated cache (GET *instant*), and to launch the cache generation process for a specific KG and depth values (POST *process*), by providing a CacheGen instance in the request body. An example request/response for cache generation can be seen in listing 7.2.
- /embedding: similarly to the cache generation embedding, it also provides a GET endpoint (*instant*) to retrieve embeddings, and a POST endpoint (*process*) that receives a list of EmbGen instances and launches the computation of the corresponding embeddings. An example request/response for embedding generation can be seen in listing 7.3.
- /agents (*instant*): it retrieves all agents stored in the system. The main purpose of this endpoint is to be used together with the testing endpoints (/tests and /tests/run).
- /experiments and /tests (*instant*): They work in a similar way, providing endpoints to insert (POST), retrieve (GET), and delete (DELETE) experiment and testing elements (either individually by id or globally) to the corresponding list. Those elements can then be run by sending a POST request to the corresponding endpoint, either /experiments/run or /tests/run (*process*). Optionally, a set of

experiment/test ids can be included as parameters in the request body to limit the scope of the training/testing.

- `/check` (*instant*): it shows the currently active processes on the system, corresponding to elements in the training/testing lists; however, it does not display their current progress. It is used mainly for debugging purposes.

Note that SpaceRL accounts for computing resources limitations and it will return a failure message to any request to a process endpoint if it determines that it will cause problems for the host machine. For instance, if there is a embedding generation endpoint running which is using the only available GPU in the system and the `/experiments/run` endpoint is invoked with configuration parameter `use_gpu = True`, the API will respond with a `BusyError`, notifying the user that there are not enough resources available.

## 7.3 Usages

adding an example of using the tool such as the one provided in the paper might be usefull to understand how it interact with itself.

The example given must be complete, how to perform it completely with the GUI and how to do it with the API by themselves as standalone applications.

In order to better understand the capabilities of SpaceRL, we selected NELL [56] as an illustrative example. NELL is a knowledge graph with over 50 million triples, which is frequently used as a resource for validation of different proposals for reasoning and completion over KGs. NELL is built automatically from web data in a continuous and mostly unsupervised fashion, meaning that there are usually a large number of missing triples, which makes it ideal to validate KG completion proposals. For this example, we use a subset of NELL built from the 995th iteration, known as NELL-995 [107].

The goal of this section is to use SpaceRL to infer missing links between concepts contained in NELL-995, together with the metric scores associated to the resulting triples, using the capabilities described in section 7.2. We assume that SpaceRL has already been correctly installed, following the instructions available in our GitHub repository [84].

We aim to demonstrate the flexibility that SpaceRL provides, which allows to invoke its functionalities either directly, as a local server, or via its API and GUI capabilities, interchangeably. To that effect, we will illustrate how to perform the different steps involved in KG completion using different strategies in each case.

Regarding the API, note that it must be deployed to make its endpoints available. To do so, we must run `API/main.py`, and wait for the messages that indicate that both the Internal Server and Client are active, and in which port number is the client
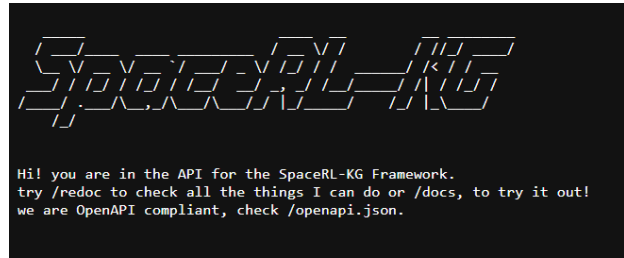
application available (cf. Figure 7.13(a) as an example of these messages).



(a) API console                                        (b) Default endpoint

**Figure 7.13:** The API being deployed in console and the webserver root

In this example, the API has been deployed to localhost, port number 8080. A request to this address and port will be responded with a welcome message that provides further instructions on how to use the tool, as seen in figure 7.13(b). The desired API endpoints are now accessible, which means that any of the existing commercial tools that automatically issue HTTP requests could be used. However, by using FastAPI, SpaceRL is able to provide a web-based, swagger-powered interface in order to access all supported endpoints at url `http://localhost:8080/docs`.

As for the GUI, it can be launched by running the file `GUI/main.py`, resulting in the main menu windows (cf. Figure 7.6).

To perform our sample KG completion, first we must provide the input KG in the expected format, and for this we will directly work on the local instance of SpaceRL. To do so, we create a new subfolder in the `/datasets` directory with the desired name, e.g., "NELL-995", and place inside a file named "graph.txt" that contains the list of triples that compose the KG. The expected file format is illustrated in listing 7.4, i.e., one $(s, r, t)$ triple per line, being each triple a sequence of three tab-separated values ($s$, $r$, and $t$). Note that this process could be performed as well using the API (endpoint `/dataset`), or the GUI (option **Datasets** in block **Folders**)

```
1 newspaper_daily_record   newspaperincity city_baltimore
2 city_baltimore   citylocatedinstate   stateorprovince_maryland
3 sportsteam_coppin_state_lady_eagles teamplaysincity city_baltimore
4 sportsteam_johns_hopkins   teamplaysincity city_baltimore
5 athlete_don_zimmerman athleteplaysinteam   sportsteam_johns_hopkins
6 politician_sheila_dixon personhasresidencein   city_baltimore
```

**Listing 7.4:** An extract of NELL dataset with the expected format.

Then, we need to provide the desired configuration for SpaceRL, including the global parameters, and the specific training and testing parameters, as described in Section 7.2.1. The values for the configuration key-value map parameters can either be set using the **Config submenu** (cf. Figure 7.7), the `/config` API endpoint, or directly by

editing the configuration file `/config.py`, as shown in listing 7.5, which also displays other possible values that the parameters can take as Python-style comments.

```python
config = {
    "available_cores": 8,
    "gpu_acceleration": True,
    "multithreaded_dist_reward": False,
    "verbose": False,
    "log_results": False,
    "debug": False,
    "print_layers": False,
    "restore_agent": False,
    "guided_reward": True,
    #"distance","terminal","embedding","shaping"
    "guided_to_compute":["terminal", "embedding"],
    "regenerate_embeddings":False,
    "normalize_embeddings":False,
    "use_LSTM": True,
    "use_episodes": False,
    "episodes":0,

    "alpha": 0.9, # [0.8-0.99] PPO prev step NN learning rate
    "gamma": 0.99, # [0.90-0.99] decay rate of past observations
    "learning_rate": 1e-3, #[1e-3, 1e-5] NN learning rate.

    "activation":'leaky_relu', # relu, prelu, leaky_relu, elu, tanh
    "regularizers":['kernel'], #"kernel", "bias", "activity"
    "algorithm": "PPO", #BASE, PPO
    "reward_type": "simple", # retropropagation, simple

    # "probability", "max"
    "action_picking_policy":"probability",

    #"max_percent", "one_hot_max", "straight"
    "reward_computation": "one_hot_max",

    "path_length":3,
    "random_seed":True,
    "seed":0
}
```

**Listing 7.5:** Configuration parameters used for the example

The next step is embedding generation, which could be omitted, since SpaceRL is able to perform it automatically. In this case, however, in order to further illustrate these functionalities in detail, all embedding representations for the NELL-995 KG will be generated. To do so, we can issue an HTTP POST request to endpoint `/embeddings`, including in the request body the parameters shown in listing 7.6.

This request will trigger the internal client to communicate with the server and initiate the embedding generation. Then, a response is sent to the API client with information regarding whether the operation has began correctly, or if any error has occurred, e.g., if the server resources are busy and the request cannot be attended. Meanwhile, the application console displays the internal client-server communication

```
1 {
2   "dataset": "NELL-995",
3   "models": [],
4   "use_gpu": true,
5   "regenerate_existing": true,
6   "normalize": true,
7   "add_inverse_path": true,
8   "fast_mode": false
9 }
```

**Listing 7.6:** The request body parameters to generate all embeddings for NELL KG.

messages, as seen in Listing 7.7.

```
1 message: "post;embedding;{'dataset': 'NELL-995', 'models': [], 'use_gpu':
      True, 'regenerate_existing': True, 'normalize': True, '
    add_inverse_path': True, 'fast_mode': False}"
2 response: "success; Embedding Calculation Launched"
```

**Listing 7.7:** Internal client-server communication for embedding generation request.

As a result, new files are added to folder `datasets/NELL-995`, namely `entities.tsv` and `relations.tsv`, which hold all entities and relations of the KG, respectively, with an assigned id. Also, a new `/embedding` subfolder is created, which holds the results of the embedding vector generation operation performed by DGL-KE for each of the required models (ComplEx, DistMult, TransE, and TransR, in the current version).

Finally, to generate the RL agents, new experiments must be added to the experiments list, via HTTP POST requests to the `/experiment` endpoint, providing the necessary parameter values in the request body. An example of a new experiment that uses TransE embeddings and 150 laps to train on the NELL-995 KG can be seen in Figure 7.8.

```
1 {
2   "name": "My_new_NELL_Agent",
3   "dataset": "NELL-995",
4   "single_relation": false,
5   "embedding": "TransE_l2",
6   "laps": 150,
7   "relation_to_train": ""
8 }
```

**Listing 7.8:** The request body parameters to generate a NELL RL Agent

In response to the former request, the server responds with an informative message (e.g., in case of success, the message returned is:

"Success": "experiment successfully added to queue.").

Some other operations that could be performed at this point are: checking the state of the queue through the same /experiment endpoint using a GET request, deleting queued elements through the DELETE request in the same endpoint, or adding more experiments to the queue by repeating the process above. For instance, if we wanted one agent for each of the embedding models we could repeat the process above to create 4 queued experiments, one per supported model.

Then, the experiments would be run by issuing a POST request to the /experiment/run endpoint, which expects a list of ids to run, or an empty list, in which case all queued experiments are run. Again, the API responds with and informative message (e.g., in case the operation was successful, the message would be :

"Success":"Experiment(s) Launched").

The agent generation operations can be computationally complex, which is why SpaceRL provides two mechanisms to track their progress:

- **Log files**, which are redirected console output from the server which is running the process, found as `logs/p_ExperimentRunner.out` and `logs/p_ExperimentRunner.err` for the general and error outputs, respectively.
- The /Check/ debug endpoint,retrieves a list of active processes in the application console, named according to the task they perform. In our example, a request to this endpoint yields the following response:

"Success;[<name='ExperimentRunner', [...], started>]"

Once the operation is complete, a new folder is created at `model/agents/My_New_NELL_Agent`, which contains the configuration options used to generate the agent, and the models that comprise it. The generated agent can be used as input to a testing process, to assess its performance and use it to infer new paths for NELL-995.

As we stated before, all the former operations could have also been invoked via our GUI. For the remainder of this Section, we will focus on this part of SpaceRL, although the testing functionality could also be invoked via the API.

Once the main GUI window is displayed, the **Config** option under the Configuration block opens the configuration submenu window (Figure 7.7). This window reflects the changes made to the key-value map in the `config.py` file, and allows for further tuning.

To proceed with testing we choose the **Setup** option under the Configuration block,

which opens the training and testing submenu (Figure 7.14). In this example, we select the Test tab, in which the generated agent can be picked out from a scrollable select input.

When an agent is selected, the available embeddings are displayed in another select input next to the first one. Additionally, we need to provide a name for the Test instance and a number of test episodes, and then click on the **add** option. The right side of the window displays the list of available tests to be run.
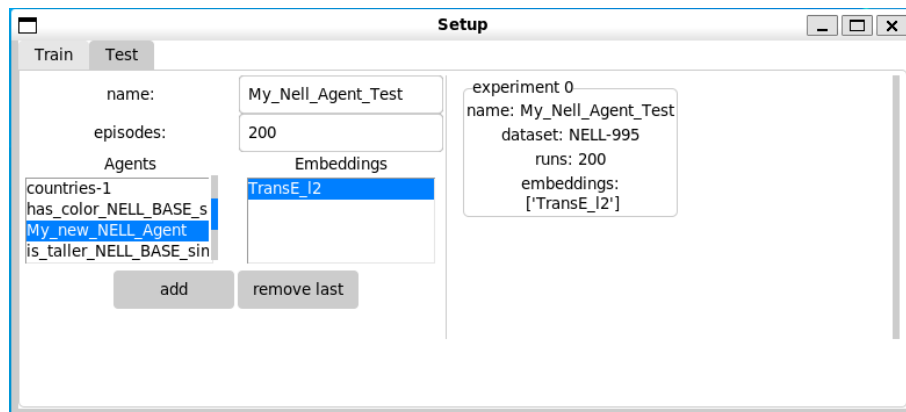


**Figure 7.14:** Testing submenu with NELL Agent being tested.

Back to the main menu window, we can use the **Test** option in the Runner block to launch as many instances of the `Trainer` class as needed, and perform up to 10,000 tests at a time for each of the elements in the test list.
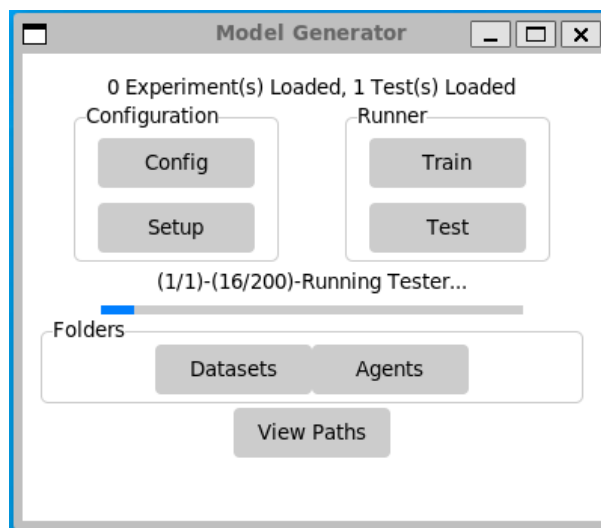


**Figure 7.15:** Main menu with running test

The main menu (Cf. Figure 7.15), now displays the progress of the current operations. Note that in the text above the progress bar, "(1/1)" indicates that only 1 element is queued for testing while "(16/200)" expresses the current and total episodes being run.

Once the execution is finished, a new folder is created in `model/data` `/results/My_NELL_Agent_Test`, containing a `metrics.csv` file with the Hits@ and MRR metric values, and a `paths.txt` file containing the inferred paths in this testing execution. Examples of the resulting metrics and the inferred new paths can be found in Listings 7.9 and 7.10, respectively.

```
1  ,,NELL-995
2  hits@1,TransE_l2,0.565
3  hits@3,TransE_l2,0.755
4  hits@5,TransE_l2,0.95
5  hits@10,TransE_l2,0.99
6  MRR,TransE_l2,0.6723
```

**Listing 7.9:** The metrics obtained from testing the Nell-995 agents for 200 episodes.

| NELL-995 | | |
|---|---|---|
| hits@1 | TransE_l2 | 0.565 |
| hits@3 | TransE_l2 | 0.755 |
| hits@5 | TransE_l2 | 0.95 |
| hits@10 | TransE_l2 | 0.99 |
| MRR | TransE_l2 | 0.6723 |

**Table 7.1:** Metrics obtained from testing the Nell-995 agents for 200 episodes.

```
1  ('concept_ceo_alan_greenspan', 'concept:topmemberoforganization', '
      concept_bank_u_s__federal_reserve', '¬concept:topmemberoforganization', '
      concept_politicianus_ben_bernanke', 'concept:worksfor', '
      concept_bank_federal_reserve')
```

**Listing 7.10:** An example of a returned path by the agent.

Observing the previously mentioned listing we can see how a typical evaluation result would appear, the first column indicates the metric that was evaluated, the second column specifies which embedding model was used, and finally, the third column displays the value of said metric. In the provided example, there is only one row for each metric, since only one embedding model (TransE) was tested.

On the other hand, in listing 7.10 we can see an example of a newly reasoned path. The path is expressed as the alternating sequence of entities and relations that must be traversed to get from the source entity until the target entity. This path provides some explainability regarding the existence of the initial query triple. For this particular example, we would have the following logic:

- Alan Greenspan is a top member of the US federal reserve.
- Ben Bernake is also a top member for the US federal reserve.
- Ben Bernake works for the federal reserve.
- Therefore, Alan Greenspan also works for the federal reserve.

- As a consequence, fact "(concept_ceo_alan_greenspan, concept:worksfor, concept_bank_federal_reserve)" should be added to the KG.

## 7.4  Support and Iterations

SpaceRL is Open Source software, open for collaboration in its GitHub repository[84] and being improved constantly. Therefore, not only does it provide benefits for companies whose business model is based on knowledge graphs and need to manipulate them; but it could also be potentially beneficial for researchers in data engineering areas who can use our functionalities as a solid support to build their own smart applications.

Finally, our tool could also aid researchers in other areas, such as biomedicine, statistics, or data science, who do not possess a computing science background that allows them to build their own software but still require this kind of tool to operate on their knowledge graphs.

We made sure that, as Open Software, SpaceRL is easily expandable by keeping the dependencies to a minimum whenever possible, providing extensive documentation and following community standards such as OpenAPI and Gymnasium.

The future steps in the life cycle of SpaceRL would be to design modular reward and policy elements that could be altered by experts to increase its reach even more, to offer a large-scale implementation to support uncoupled operations and to increase the level of the documentation offered

## 7.5  Summary

In this chapter, we have described the SpaceRL framework, and end-to-end accessible tool for knowledge graph reasoning, that allows for flexible operation and expansion. It is a powerful and flexible tool to help with Knowledge Graph problems while also being expandable and highly customizable, and potentially used for the development of novel and improved reasoning applications over KGs.

# Part IV

# Final Remarks

# Chapter 8

# Conclusions

*"Some peoples only exercise is jumping to conclusions."*

*— Author Unknown.*

In this dissertation, we have presented SpaceRL a framework offering a set of reinforcement learning tools tailored towards Knowledge Graph completion and reasoning; a versatile tool with different available interfaces, a visualization tool to graphically display the results, a GUI that provides access to less experienced users, and a REST API that enables for it to be operated as a MLaaS.

SpaceRL was designed not only as a tool for final users but also as a base for other developers to create their own custom tools, which can be offered as a service to a third party through its API complying with the openAPI convention. This might appeal to companies who wish to either serve or consume the capabilities offered.

Other existing proposals have merely scratched the surface when it comes to reward functions, applying unanimously terminal-based reward functions with minimal modifications and the backpropagation-focused REINFORCE algorithm in tandem.

Our novel alternative consists of a new set of reward functions and the application of RL algorithms whose potential remained unexplored. Our reward functions seek to use graph-specific information that is available before reaching the end of an episode: the distance to the answer node, and the semantic similarity to it computed from node embeddings. The implemented technique makes use of Proximal Policy Optimization and the Actor-Critic paradigm, resulting in faster training.

Both the new reward functions and policies have resulted in improvements over

the state-of-the-art standard practices, particularly when using embedding-based reward functions on five widely used datasets. These results should motivate the development and evaluation of more variants of these aspects since there is a margin for improvement. Therefore, two trends of future work could be developed: 1) evaluating existing context-independent RL techniques, which are often already implemented by existing libraries but mainly remain untested in this context; 2) implementing new reward functions that make use of additional information in the graph, e.g. node attributes, which provide additional rich data.

# Bibliography

[1] M. M. Afsar, T. Crump, and B. Far. Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38, 2022.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *international semantic web conference*, pages 722–735. Springer, 2007.

[3] I. Balažević, C. Allen, and T. M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590*, 2019.

[4] K. Bollacker, R. Cook, and P. Tufts. Freebase: A shared database of structured general human knowledge. In *AAAI*, volume 7, pages 1962–1963, 2007.

[5] A. Bordes and E. Gabrilovich. Constructing and mining web-scale knowledge graphs: Kdd 2014 tutorial. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1967–1967, 2014.

[6] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[7] A. Borrego, D. Ayala, I. Hernández, C. R. Rivero, and D. Ruiz. Generating rules to filter candidate triples for their correctness checking by knowledge graph completion techniques. In M. Kejriwal, P. A. Szekely, and R. Troncy, editors, *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019*, pages 115–122. ACM, 2019. doi: 10.1145/3360901.3364418. URL https://doi.org/10.1145/3360901.3364418.

[8] A. Borrego, D. Ayala, I. Hernandez, C. R. Rivero, and D. Ruiz. Cafe: Knowledge graph completion using neighborhood-aware features. *Eng. Appl. Artif. Intell.*, p.p., 2021.

[9] F. Che, D. Zhang, J. Tao, M. Niu, and B. Zhao. Parame: Regarding neural network parameters as relation embeddings for knowledge graph completion.

In *Proceedings of the AAAI Conference on Artificial Intelligence,* volume 34, pages 2774–2781, 2020.

[10] Z. Chen, Y. Wang, B. Zhao, J. Cheng, X. Zhao, and Z. Duan. Knowledge graph completion: A review. *Ieee Access*, 8:192435–192456, 2020.

[11] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *ACM Sigmod Record*, 35(3):34–41, 2006.

[12] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[13] H. Cui, T. Peng, R. Han, B. Zhu, H. Bi, and L. Liu. Reinforcement learning with dynamic completion for answering multi-hop questions over incomplete knowledge graph. *Information Processing & Management*, 60(3):103283, 2023.

[14] H. Cui, T. Peng, F. Xiao, J. Han, R. Han, and L. Liu. Incorporating anticipation embedding into reinforcement learning framework for multi-hop knowledge graph question answering. *Information Sciences*, 619:745–761, 2023.

[15] Y. Dai, S. Wang, N. N. Xiong, and W. Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

[16] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017.

[17] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[18] T. Dettmers, M. Pasquale, S. Pontus, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pages 1811–1818, February 2018. URL `https://arxiv.org/abs/1707.01476`.

[19] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, pages 100–110, 2004.

[20] F. Such, Vashisht Madhavan, Rosanne Liu, and J. Lehman. An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. *International Joint Conference on Artificial Intelligence*, 2018.

[21] Farama Foundation. Gymnasium. `https://github.com/Farama-Foundation/Gymnasium`, 2023. [Accessed July 2023].

[22] FastAPI. FastAPI. `https://github.com/tiangolo/fastapi`, 2023. [Accessed July 2023].

[23] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni. Robotic arm control and task training through deep reinforcement learning. In *International Conference on Intelligent Autonomous Systems*, pages 532–550. Springer, 2021.

[24] M. H. Gad-Elrab, D. Stepanova, T.-K. Tran, H. Adel, and G. Weikum. Excut: Explainable embedding-based clustering over knowledge graphs. In *International Semantic Web Conference*, pages 218–237. Springer, 2020.

[25] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730, 2015. doi: 10.1007/s00778-015-0394-1.

[26] O.-E. Ganea and T. Hofmann. Deep joint entity disambiguation with local neural attention. *arXiv preprint arXiv:1704.04920*, 2017.

[27] M. Gardner and T. Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. In *EMNLP*, pages 1488–1498. The Association for Computational Linguistics, 2015.

[28] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[31] Y. Hirose, M. Shimbo, and T. Watanabe. Transductive data augmentation with relational path rule mining for knowledge graph embedding. In *2021 IEEE International Conference on Big Knowledge (ICBK)*, pages 377–384. IEEE, 2021.

[32] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[33] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[34] B. Jia, C. Dong, Z. Chen, K.-C. Chang, N. Sullivan, and G. Chen. Pattern discovery and anomaly detection via knowledge graph. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 2392–2399. IEEE, 2018.

[35] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[36] Keras. Keras. `https://github.com/keras-team/keras`, 2023. [Accessed July 2023].

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[39] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[40] K. Kolthoff and A. Dutta. Semantic relation composition in large scale knowledge bases. In *LD4IE@ISWC*, volume 1467, pages 34–47, 2015.

[41] N. Lao, T. Mitchell, and W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 529–539, 2011.

[42] P. Le and I. Titov. Improving entity linking by modeling latent relations between mentions. *arXiv preprint arXiv:1804.10637*, 2018.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[44] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2): 167–195, 2015.

[45] X. V. Lin, R. Socher, and C. Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

[46] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[47] T. Lindner, A. Milecki, and D. Wyrwał. Positioning of the robotic arm using different reinforcement learning algorithms. *International Journal of Control, Automation and Systems*, 19:1661–1676, 2021.

[48] X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[49] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[50] S. Manchanda. Metapath-guided data-augmentation for knowledge graphs. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 4175–4179, 2023.

[51] B. Marr. How much data do we create every day? the mind-blowing stats everyone should read. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*, Jul 2021. URL https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/.

[52] J. L. Martinez-Rodriguez, A. Hogan, and I. Lopez-Arevalo. Information extraction meets the semantic web: a survey. *Semantic Web*, 11(2):255–335, 2020.

[53] S. Mazumder and B. Liu. Context-aware path ranking for knowledge base completion. In *IJCAI*, pages 1195–1201. AAAI Press, 2017. doi: 10.24963/ijcai.2017/166.

[54] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[55] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[56] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. *Commun. ACM*, 61(5):103–115, 2018. doi: 10.1145/3191513.

[57] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[58] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[59] NetworkX. NetworkX. https://github.com/networkx/networkx, 2023. [Accessed July 2023].

[60] D. N. Nicholson and C. S. Greene. Constructing knowledge graphs and their biomedical applications. *Computational and structural biotechnology journal*, 18: 1414–1428, 2020.

[61] M. Nickel, V. Tresp, H.-P. Kriegel, et al. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 3104482–3104584, 2011.

[62] OpenAI. OpenAI Gym. `https://github.com/openai/gym`, 2023. [Accessed July 2023].

[63] OpenAPI. OpenAPI. `https://github.com/OAI/OpenAPI-Specification`, 2023. [Accessed July 2023].

[64] X. Peng, G. Chen, C. Lin, and M. Stevenson. Highly efficient knowledge graph embedding learning with orthogonal procrustes analysis. *arXiv preprint arXiv:2104.04676*, 2021.

[65] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[66] PyGame. PyGame. `https://github.com/pygame/pygame`, 2023. [Accessed July 2023].

[67] X. Ren, W. He, M. Qu, C. R. Voss, H. Ji, and J. Han. Label noise reduction in entity typing by heterogeneous partial-label embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1825–1834, 2016.

[68] M. Ribeiro, K. Grolinger, and M. A. Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902, 2015. doi: 10.1109/ICMLA.2015.152.

[69] P. Rojanavasu, P. Srinil, and O. Pinngern. New recommendation system using reinforcement learning. *Special Issue of the Intl. J. Computer, the Internet and Management*, 13(SP 3), 2005.

[70] A. Saeedi, E. Peukert, and E. Rahm. Using link features for entity clustering in knowledge graphs. In *European Semantic Web Conference*, pages 576–592. Springer, 2018.

[71] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. doi: 10.1147/rd.33.0210.

[72] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

[73] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[74] SDL team. SDL Kit. `https://www.libsdl.org/`, 2023. [Accessed July 2023].

[75] A. Senaratne, P. G. Omran, G. Williams, and P. Christen. Unsupervised anomaly

detection in knowledge graphs. In *Proceedings of the 10th International Joint Conference on Knowledge Graphs*, pages 161–165, 2021.

[76] T. Shen, F. Zhang, and J. Cheng. A comprehensive overview of knowledge graph completion. *Knowledge-Based Systems*, page 109597, 2022.

[77] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

[78] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[79] B. F. Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.

[80] B. F. Skinner. *Contingencies of reinforcement: A theoretical analysis*, volume 3. BF Skinner Foundation, 2014.

[81] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013.

[82] F. Sola, D. Ayala, R. Ayala, I. Hernández, C. R. Rivero, and D. Ruiz. Aynext-tools for streamlining the evaluation of link prediction techniques. *SoftwareX*, 23:101474, 2023. doi: https://doi.org/10.1016/j.softx.2023.101474. URL `https://www.sciencedirect.com/science/article/pii/S235271102300170X`.

[83] F. Sola, D. Ayala, I. Hernández, and D. Ruiz. Deep embeddings and graph neural networks: using context to improve domain-independent predictions. *Applied Intelligence*, 53(19):22415–22428, 2023.

[84] SpaceRL. SpaceRL. `https://github.com/DEAL-US/SpaceRL-KG`, 2023. [Accessed July 2023].

[85] T. Steiner, R. Verborgh, R. Troncy, J. Gabarro, and R. Van de Walle. Adding realtime coverage to the google knowledge graph. In *11th International Semantic Web Conference (ISWC 2012)*, volume 914, pages 65–68. Citeseer, 2012.

[86] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

[87] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[88] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[89] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[90] Swagger. Swagger. `https://github.com/swagger-api/swagger-ui`, 2023. [Accessed July 2023].

[91] X. Tang, Y. Chen, X. Li, J. Liu, and Z. Ying. A reinforcement learning approach to personalized learning recommendation systems. *British Journal of Mathematical and Statistical Psychology*, 72(1):108–135, 2019.

[92] TensorFlow. TensorFlow. `https://github.com/tensorflow/tensorflow`, 2023. [Accessed July 2023].

[93] G. Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[94] E. L. Thorndike. The fundamentals of learning. *Self Published.*, 1932.

[95] P. Tiwari, H. Zhu, and H. M. Pandey. Dapath: Distance-aware knowledge graph reasoning based on deep reinforcement learning. *Neural Networks*, 135:1–12, 2021.

[96] H. N. Tran and A. Takasu. Multi-partition embedding interaction with block term format for knowledge graph completion. *arXiv preprint arXiv:2006.16365*, 2020.

[97] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[98] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[99] Uvicorn. Uvicorn. `https://github.com/encode/uvicorn`, 2023. [Accessed July 2023].

[100] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3009–3016, 2020.

[101] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[102] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[103] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

[104] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[105] Y. Xian, Z. Fu, S. Muthukrishnan, G. De Melo, and Y. Zhang. Reinforcement knowledge graph reasoning for explainable recommendation. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 285–294, 2019.

[106] X. Xin, A. Karatzoglou, I. Arapakis, and J. M. Jose. Self-supervised reinforcement learning for recommender systems. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 931–940, 2020.

[107] W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017.

[108] P. Xu and D. Barbosa. Neural fine-grained entity type classification with hierarchy-aware loss. *arXiv preprint arXiv:1803.03378*, 2018.

[109] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):40–51, 2006.

[110] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[111] A. Yates, M. Banko, M. Broadhead, M. J. Cafarella, O. Etzioni, and S. Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 25–26, 2007.

[112] Ze Yang, and Liwei Wang. Learning to navigate for fine-grained classification. *European Conference on Computer Vision*, 2018.

[113] D. Zeng, K. Liu, Y. Chen, and J. Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1753–1762, 2015.

[114] D. Zheng, X. Song, C. Ma, Z. Tan, Z. Ye, J. Dong, H. Xiong, Z. Zhang, and G. Karypis. Dgl-ke: Training knowledge graph embeddings at scale.

In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020.

[115] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 207–212, 2016.