POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An experimental analysis of Link Prediction methods over Microservices Knowledge Graphs

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Gianluca Ruberto**

Student ID: 994175
Advisor: Prof. Davide Martinenghi

Academic Year: 2022-23

Abstract

Graphs are a powerful way to represent data. They can be seen as a collection of objects (nodes) and the relationships between them (edges or links). The power of this structure has its intrinsic value in the relationship between data points that can even provide more information than the data properties. An important type of graph is Knowledge Graphs in which each node and edge has a type associated. Often graph data is incomplete and in this case, it is not possible to retrieve useful information. Link prediction, also known as knowledge graph completion, is the task of inferring if there are missing edges or nodes in a graph. Models of different types, including Machine Learning-based, Rule-based, and Neural Network-based models have been developed to address this problem. The goal of this research is to understand how link prediction methods perform in a real use-case scenario. Therefore, multiple models have been compared on different accuracy metrics and production case requirements on a microservice tracing dataset. Models have been trained and tested on two different knowledge graphs obtained from the data, one that takes into account the temporal information, and the other that does not. Moreover, the prediction of the models has been evaluated with what is usually done in the literature, and also mimicking a real use-case scenario. The comparison showed that too complex models cannot be used when the time, at training, and/or inference phase, is critical. The best model for traditional prediction has been RotatE which usually doubled the score of the second-best model. Considering the use-case scenario, RotatE was tied with QuatE, which required a lot more time for training and predicting. They scored 20% to 40% better than the third-best performing model, depending on the case. Moreover, most of the models required less than a millisecond for predicting a triplet, with NodePiece that was the fastest, beating ConvE by a 4% margin. For the training time, NodePiece beats AnyBURL by 40%. Considering the memory usage, again NodePiece is the best, by an order of magnitude of at least 10 when compared to most of the other models. RotatE has been considered the best model overall because it had the best accuracy and an above-average performance on the other requirements. Additionally, a simulation of the integration of RotatE with a dynamic sampling tracing tool has been carried out, showing similar results to the ones previously obtained. Lastly, a thorough analysis of the

results and suggestions for future work are presented.

Keywords: Knowledge Graphs, Link Prediction, Machine Learning, Microservice Tracing

Abstract in lingua italiana

I grafi sono un approccio efficace per rappresentare i dati. Possono essere visti come una collezione di oggetti (nodi) e le relazioni tra di essi (archi o collegamenti). L'efficacia di questa struttura risiede nel valore intrinseco delle relazioni tra i punti dati, che possono fornire informazioni anche più significative delle proprietà dei dati stessi. Un tipo importante di grafo è il Knowledge Graph, in cui a ciascun nodo e arco è associato un tipo. Spesso i dati del grafo sono incompleti e, in questo caso, non è possibile recuperare informazioni utili. La predizione dei collegamenti, nota anche come completamento del knowledge graph, consiste nel compito di inferire se ci sono archi o nodi mancanti in un grafo. Sono stati sviluppati modelli di diversi tipi, tra cui modelli basati su apprendimento automatico, basati su regole e basati su reti neurali, per affrontare questo problema. L'obiettivo di questa ricerca è comprendere come i metodi di previsione dei collegamenti si comportano in uno scenario di utilizzo reale. Pertanto, sono stati confrontati più modelli su diverse metriche di accuratezza e requisiti relativi a production-level use-cases su un dataset di tracciamento di microservizi. I modelli sono stati allenati e testati su due diversi knowledge graph ottenuti dalla modellazione dei dati, uno che tiene conto delle informazioni temporali e l'altro che non le tiene in considerazione. Inoltre, le predizioni dei modelli sono state valutate sia come di consueto nella letteratura, sia simulando uno scenario di utilizzo reale. Il confronto ha mostrato che modelli troppo complessi non possono essere utilizzati quando il tempo durante la fase di allenamento e/o inferenza è critico. Il miglior modello per la predizione tradizionale è stato RotatE, che di solito ha raddoppiato il punteggio del secondo miglior modello. Considerando la predizione nello use-case, RotatE è stato pari a QuatE, che richiedeva molto più tempo per l'allenamento e la previsione. Hanno ottenuto punteggi migliori del 20% al 40% rispetto al terzo miglior modello, a seconda del caso. Inoltre, la maggior parte dei modelli ha richiesto meno di un millisecondo per prevedere una tripletta, con NodePiece che è stato il più veloce, superando ConvE di un margine del 4%. Considernado il tempo di training, NodePiece batte AnyBURL del 40%. Considerando l'utilizzo della memoria, ancora una volta NodePiece è il migliore, con un ordine di grandezza almeno 10 volte superiore rispetto alla maggior parte degli altri modelli. RotatE è stato considerato il miglior modello complessivo perché

aveva la migliore accuratezza e una prestazione sopra la media sugli altri requisiti. Inoltre, è stata effettuata una simulazione dell'integrazione di RotatE con un tool di tracciamento di campionamento dinamico, mostrando risultati simili a quelli ottenuti in precedenza. Infine, vengono presentati un'analisi approfondita dei risultati e suggerimenti per i lavori futuri.

Parole chiave: Knowledge Graphs, Link Prediction, Machine Learning, Microservice Tracing

Contents

1 | Introduction

This chapter provides a general introduction to the research area of the degree project. It gives an overview of the background and its main topics: Knowledge Graphs, Link Prediction, and Microservices. It explains why these fields are very interesting, as knowledge graphs have been applied in various fields such as recommender systems [64], drug side-effect discovery [70], and search engines like Google Knowledge Graph [20], while the microservice architecture is also attractive since it is used by many tech giants including Microsoft, Google, Amazon [38], and Netflix [54]. It presents the problem addressed in these areas and explains the objectives of the thesis. In particular, it describes how requirements of low inference time, training time, and memory usage are critical for both knowledge graphs link prediction and microservice-based applications. Therefore it presents how using link prediction methods to integrate a tracing tool such as Jaeger [23], seems an optimal case study since link prediction methods may solve the performance degradation problem caused by tracing tools. Moreover, it explores the ethics and sustainability aspects of the thesis, showing that achieving good results on the timing requirements can have a positive impact on the environment. Furthermore, the methodology followed for the comparison of the chosen models and the main delimitations are presented. Lastly, the last section describes the outline of the following chapter of the thesis.

1.1. Background

The scope of the thesis lies in the areas of Knowledge Graphs Link Prediction, and Microservices which will be explored in the following subsections.

1.1.1. Knowledge Graphs

Data can be represented through graphs. A graph $G = (V,E)$ is a set of nodes V and a set of edges E between these nodes [17]. When data is represented in this way, connections between entities may be more meaningful than entities' features [17]. Following this direction, Knowledge Graphs (KGs) are able to achieve more complex representations by

having different types of edges. KGs are useful for modelling heterogeneous interactions and therefore have a wide range of applications, including recommender systems [64], drug side-effect discovery [70], and search engines like Google Knowledge Graph [20].

1.1.2. Link Prediction

It is known that knowledge graphs are often incomplete [29]. The most famous example is Freebase [6], in which 71% of people have no known place of birth, and 75% have no known nationality [10].

Link Prediction or *Knowledge Graph Completion* is the task of solving this problem by inferring missing edges to the graph.

Several approaches have been used to solve this problem. Traditional *topology-based* algorithms look at the graph structure to see if two nodes are likely to be connected through an edge [34].

Logical Rules can also be applied to this problem. Approaches like AnyBURL [35] showed that it is possible to use "if-then" rules, in the form of implication as shown in Equation 1.1, also for large knowledge graphs :

$$X \rightarrow Y \text{ (if X then Y)} \tag{1.1}$$

Machine Learning approaches focused on finding the *embedding*, a good representation of the nodes and edges in a lower dimensional space. Meaningful approaches are TransE [7] that modelled relations as translations in the learnt embedding space, DistMult [63] that uses cosine similarity of the embeddings for making predictions, ComplEx [55] that, as the name suggests, uses complex embeddings to model different types of relations. Modern approaches in this area are RotatE [52] which represents nodes as complex vectors and edges as rotations in complex vector space, QuatE [67] that uses quaternion hypercomplex as embedding space, and TuckER [4] which is a tensor method based on Tucker decomposition [56].

Neural Networks (NNs) have been applied to graphs, they are used to create deep embeddings. ConvE [9] is a Convolutional Neural Network (CNN) that concatenates entities and relations to use a 2D Convolutional layer. R-GCN [48] is a Recurrent Graph Neural Network (RecGNN) which has been specifically developed for Knowledge Graphs. It is based on the *message-passing* framework. NodePiece [14] uses tokenization to learn a vocabulary and therefore it is suitable for inductive link prediction, which differently from

transductive link prediction, allows unseen entities at the inference phase.

1.1.3. Microservices

Current large-scale applications are developed using the microservice architecture. A microservice is a small application that can be developed and managed independently [54]. Multiple microservices interact with each other to create the application. They are used by many tech giants including Microsoft, Google, Amazon [38], and Netflix [54]. Nevertheless, this architecture is not easy to maintain. For this reason, tracing tools such as Dapper [50], Jaeger [23], Zipkin [41], and OpenTelemetry [40] have been developed to create a microservice tracing graph. A microservice tracing graph is used to track the user request path inside the system. Requests are monitored by attaching relevant contextual metadata along with them during their execution [49], allowing developers to analyse their applications. Common tasks are improving performance by bottleneck/critical path detection, request time measurement, root cause analysis, and automatic scaling.

Still, this approach has a big drawback in performance degradation since an overhead on the messages exchanged between microservices is required to create such a graph. Link prediction methods may be a way to solve this problem.

Why Microservices?

To the best of my knowledge, previous studies that compared link prediction methods, used often the same dataset like FB15k [7], WN18 [7], YAGO3-10 [9], etc, and focused mainly on accuracy. They usually did not take into account inference time, training time, and memory usage, which are requirements that have been identified according to open problems pointed out from previous studies [19, 38, 45], but are also seen as crucial challenges in microservice-based application [45, 66, 68]. In particular:

- Inference time is considered a critical requirement for applications that make use of microservices graphs, such as CRISP [68] that performs critical path analysis, or DeepTraLog [66] that perform anomaly detection.
- Training time is an essential requirement for the same reasons.
- Memory usage has been considered fundamental by applications that use microservices knowledge graphs to perform root cause analysis [45].

Therefore this work will focus on knowledge graph link prediction in the microservices area since it is an optimal use case to explore the requirements stated before.

Nevertheless, microservice graphs are a bit different from the ones usually explored in the literature. Famous KGs used in the literature such as FB15k [7], WN18 [15], and YAGO3-10 [33] are very large in the number of entities, relations, and triplets. Having such data, models are trained on a big single graph where some links have been removed. Instead, microservices call graphs are completely different, since they are multiple small graphs corresponding to the different traces. Consequently, it is necessary to develop a different approach to understand how models perform in a real use-case scenario.

1.2. Problem

As previously stated, past research focused mainly on the accuracy performance of link prediction models and did not focus on other metrics that are considered equally important [19, 38, 45]. Therefore it is not clear how the above-mentioned models will perform in a real use-case scenario where there are also other requirements such as training time, inference time, and memory usage are relevant. Moreover, a comprehensive analysis of the microservice use case has not been performed yet.

1.2.1. Related Work

This section includes both comparison or survey papers on Knowledge Graph Link Prediction and studies that apply graph theory to different domains including antiviral drugs, and microservices. However, this work is unique, since to the best of my knowledge, there is no previous research that applied link prediction techniques to incomplete microservice graphs.

Rossi et al. [29] carried out a comprehensive comparison of several Machine Learning (ML) and NNs link prediction methods. The paper is well-written and very detailed. Moreover, it classifies models into different categories. It compares their performance on known datasets already used in research, therefore it does not analyse their behaviour in a real use case. Moreover, it does not take into account newer models like QuatE [67], and NodePiece [14]. Even though the training time has been compared, it does not seem to be a fair comparison. In particular, for different methods, different optimizers and different loss functions have been used, moreover, the early stopping technique has not been used, and therefore the training time depends on the arbitrary choice of the training epochs.

MDistMult [59] proposes a new model to perform link prediction on antiviral drugs knowledge graphs. They also compare with state-of-the-art models including RotatE [52],

TuckER [4], QuatE [67], ConvE [9], etc. It is an interesting research since it shows how Knowledge Graph (KG) link prediction can be applied to an important domain. However, it does not take into account metrics other than accuracy, differently from this thesis.

Qiu et al. [45] modelled the microservice tracing graph as a knowledge graph and then applied a causal search algorithm for the root cause analysis problem. Their work, however, is different from the one in this thesis since they use as nodes not only microservices, but also other software such as middleware, databases, containers, etc, and also hardware such as hard disks, virtual machines, clusters, etc. In this work, instead, the nodes will be solely microservices. Therefore also the relations that they use are different. Moreover, they do not apply link prediction to reconstruct the graph, as it happens in this thesis, but they build it from logs and traces.

B-MEG [3] is an approach to find bottlenecks in microservices graphs. The graph is made by RPC between microservices. The nodes of the call graph are RPCs of microservices and the edges correspond to an invocation of RPC from an upstream microservice to a downstream microservice. It performs graph classification on a trace to understand if it is anomalous, and then it performs node classification on the trace to understand if it is a bottleneck. However, it applies Graph Neural Networks (GNNs) but it does not model the graph as a knowledge graph.

It appears obvious that research focused mainly on the accuracy performance of link prediction models, and therefore it is not clear how the above-mentioned models will perform in a real use-case scenario where there are also other requirements. Furthermore, when the previously mentioned methods have been proposed they did not take into account the computational complexity or the memory usage, like for R-GCN [48], and ConvE [9], and if they did like for QuatE[67], and TuckER [4], they just gave a mathematical bound that does not give meaningful information.

For these reasons, this project aims to address these problems.

1.3. Purpose

The thesis project aims to compare different link prediction models on microservices knowledge graphs. The comparison will be performed according to accuracy, but also on metrics specific to the use case previously mentioned in Section 1.2. To summarize, the research question would be the following:

How do state-of-the-art link prediction methods compare when it comes to accuracy, in-

ference time, training time, and memory usage?

1.4. Goal

The goal of the degree project is to understand how link prediction methods perform in a real use-case scenario where also other metrics than accuracy are used.
To do so, the following steps are carried out:

1. Identification of key requirements in the knowledge graph link prediction area;
2. Identification of state-of-the-art link prediction methods;
3. Identification of an appropriate type of data (microservice traces);
4. Defining appropriate ways to build the knowledge graph;
5. Train the selected models;
6. Measure the accuracy performance from a Data Science perspective;
7. Understand how to measure the performance from a use-case perspective, and measure it;
8. As an additional step, use the best model to simulate the performance of a real-case scenario.

1.5. Benefits, Ethics and Sustainability

All the areas that model data as KG will be the beneficiaries of this project. In particular, the analysis of the timing constraint will have a positive effect on the sustainability side. According to the work of Patterson et al [43], the carbon emission of a machine learning model is directly proportional to the time required at training and prediction phase. Therefore, by performing a comparison using the same configuration, it is possible to have a relative comparison of the carbon emission of the models. Nevertheless, the models used are small, so their carbon emission is not expected to be excessive. However, understanding if the use of machine learning models is more energy efficient than the use of tracing sampling tools is outside of the scope of this thesis.

Considering the ethics aspect, there are no evident concerns, since the data used does not contain personal information.

1.6. Methodology

For this thesis, following the framework of Håkansson [16], it has been decided to perform a quantitative research with a deductive approach and an experimental strategy. However, the final comparison followed a qualitative approach to choose the best model.

The Knowledge Graph has been obtained from the data by using two different modelling approaches: the *static* modelling approach and the *IO* modelling approach. In the former scenario, the selected models will be compared on link prediction both in *transductive* and *inductive* settings. Moreover, in addition to the usual comparison that is done in the literature, the model will be compared also in a way that simulates a real use-case scenario prediction. Therefore there will be two different prediction scopes, as explained in Section 4.1.

As a result, it will be possible to understand how the models perform on the requirements mentioned in Section 1.2 and how they would perform in a real production-level use case.

1.7. Delimitations

There are multiple delimitations that affected this research. A delimitation is the implementation of methods on Python libraries that can address the KG link prediction task since the goal of this thesis is to compare the methods and not to develop new ones. Similarly, due to obvious time limitations, it has not been possible to perform hyperparameter tuning and therefore find the best configuration for all the models. Therefore, the hyperparameters have been chosen as the default ones in the Python implementation or the ones suggested by their authors. Moreover, some features of the data, such as timestamps have not been used, since the chosen class of models does not support them. For the same reason, it has not been possible to discuss online training or dynamic graph ingestion.

1.8. Outline

The following chapters will go more in-depth into the research. In particular, Chapter 2 describes in detail graph theory, link prediction methods, and the challenges of this research area. Chapter 3 explains the research method, and Chapter 4 shows its application. Chapter 5 presents the artefacts. Chapter 6 describes the experimental results, and Chapter 7 presents the conclusion.

2 | Theoretical Background

In this chapter, I talk about the theoretical background on which the thesis is founded.

2.1. Graph Theory

This section deepens into graph theory, presenting what graphs are and one particular type called *Knowledge Graphs*.

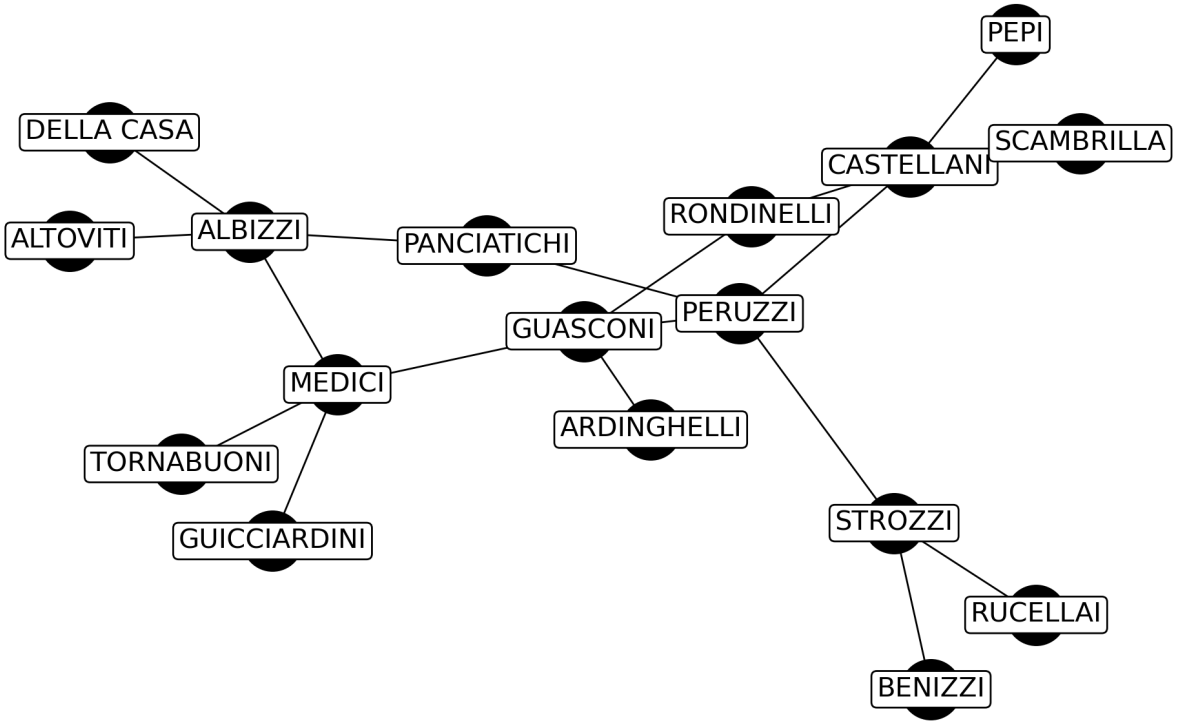
2.1.1. Graphs

From a mathematical point of view, a graph $G = (V,E)$ is a set of nodes V and a set of edges E between these nodes [17]. In the literature, graphs are often called networks, nodes are called entities or vertices, and edges are called relations or links. In this work, these terms will be used interchangeably.

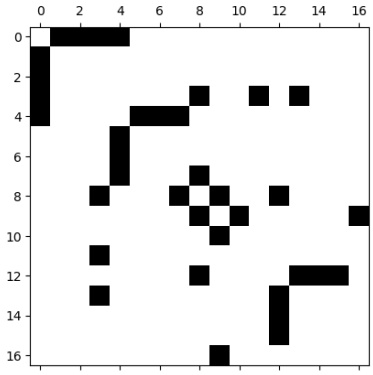
Graphs can be represented in two ways: Using the *Adjacency Matrix* or using the *Adjacency List* [5]. The *adjacency matrix* $A \in R^{|V|\times|V|}$, is a matrix in which a 1 in position i,j means that there is an edge between node i and node j. The *adjacency list*, instead, is a list of elements where each of them is a pair of nodes (i, j) that are connected through an edge. The two representations are interchangeable, and the choice of which to use depends on the use case. Both representations can be seen in Figure 2.1.

There are various types of graphs based on the characteristics of nodes and edges. Edges can be *directed*, which means that they can be traversed according to one direction or *undirected*, which means that they can be traversed in both directions. Edges can be *weighted*, or there can be *multiple edges* between two nodes.

Graphs are a powerful way to represent data, The power of this structure has its intrinsic value in the relationship between data points that can even provide more information than the data properties [17]. In some cases, such as the internet or social networks, a graph representation arises naturally from the data structure. In other cases, like citation networks, using a graph structure is still useful to get important insight.



(a) Graph.



(b) Adjacency matrix.

```
MEDICI GUICCIARDINI
MEDICI TORNABUONI
MEDICI GUASCONI
MEDICI ALBIZZI
GUASCONI PERUZZI
GUASCONI ARDINGHELLI
.
.
ALBIZZI DELLA CASA
ALBIZZI PANCIATICHI
```

(c) Adjacency list.

Figure 2.1: Marriage graph of aristocratic Florentine families from the 15 th century [42] (drawn using newtworkx).

Centrality measures are fundamental for topology-based methods, which will be seen in Section 2.2.1. They answer to the question "Which are the most important or central vertices in a network?" [36]. A simple example is *node degree* which is the number of edges that a vertex has. The degree is occasionally called *degree centrality*, and despite its simplicity, it can be enlightening. In a social network, it is reasonable to think that entities with a high *degree centrality* might have more influence, more access to information, and more prestige than those that have a lower number of links [36]. A famous example is the

marriage graph of aristocratic Florentine families from the 15$^{\text{th}}$ century [42] which can be seen in Figure 2.1.

Analysing data represented as a graph, it is possible to discover that the Medici family had a high *centrality* value and therefore the following prosperity is not a surprise [22]. There are several centrality measures and they do not give always the same results. However, their description is not in the scope of this thesis.

Graphs mentioned so far are called *simple graphs* [17]. However, multiple types of graphs. *Multi-relational* graphs or *KGs* have edges of different types [17]. *Heterogeneous* graphs have also nodes of multiple types [17], while *Multiplex* graphs assume that a graph can be decomposed in a set of k layers where each node belongs to a single layer and each layer represents a unique relation [17]. The focus of this research is on KGs. For this reason, the next section is dedicated to describing them in more detail.

2.1.2. Knowledge Graphs

Knowledge Graphs or Multi-Relational Graphs are a particular type of graph where edges can have types [17]. As a result, the adjacency matrix A is now a multi-dimensional matrix, $A \in R^{|V|\times|V|\times|T|}$, where $/T/$ is the total number of edge types. Thanks to this improvement, knowledge graphs can be used to represent a more complex interaction between data points. In particular, knowledge graphs are able to represent those data in which connections have different meanings, and some types of connections are not possible between all nodes. Considering the previous example with the Florentine families of the 15 $^{\text{th}}$ century, it is possible to have a more rich representation when taking into account other relations than the marriage. The result of considering also trade and real estate partnership can be seen in Figure 2.2. Examples of the application of KGs are drug side-effect discovery [70] where each side-effect is a link of a different type, and search engines like Google Knowledge Graph [20].

2.2. Link Prediction

It is known that knowledge graphs are often incomplete [29]. The most famous example is Freebase [6], in which 71% of people have no known place of birth, and 75% have no known nationality [10].

Link Prediction or *Knowledge Graph Completion* is the task of solving this problem by inferring missing edges to the graph. It can be split into two subtasks: *entity prediction*, and *relation prediction* [58]. Furthermore, entity prediction is called in two different ways

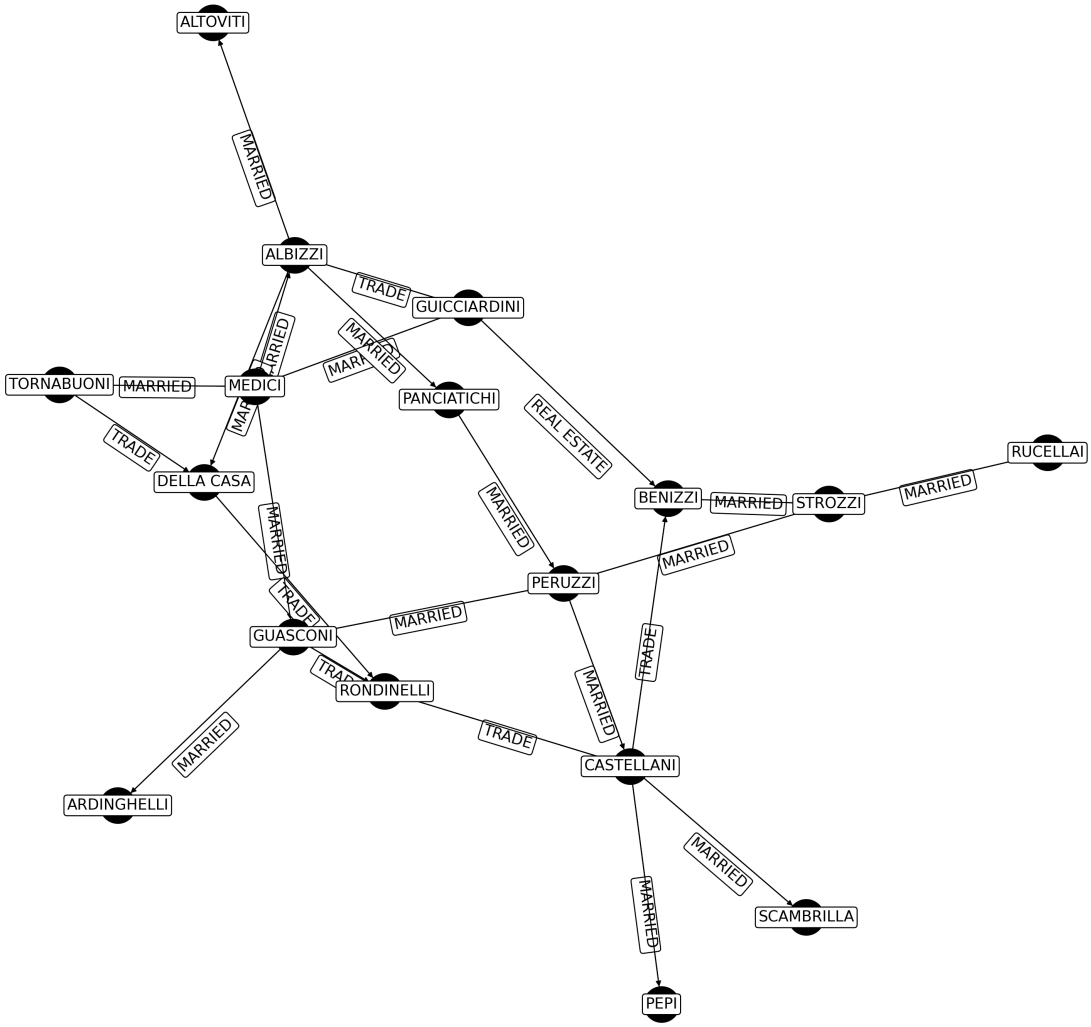


Figure 2.2: Relationship knowledge graph of aristocratic Florentine families from the 15 $^{\text{th}}$ century [42] (drawn using newtworkx).

depending on which entity we want to predict [58]. In particular, given a triplet fact of a KG, $\langle h, r, t \rangle$, where h is the head entity, r is the relation, and t is the tail entity, we have:

- *head prediction*, when the head entity is missing: $\langle ?, r, t \rangle$
- *tail prediction*, when the tail entity is missing: $\langle h, r, ? \rangle$
- *relation prediction*, when the relation is missing: $\langle h, ?, t \rangle$

More generally, the known entity is called *source entity*, and the missing one *target entity*

[29]. In particular, the scope of a model is to define a scoring function $\phi(h, r, t)$ that gives the plausibility of a given fact. [29].

In the past years, various approaches have been developed over time to tackle this problem. They can be split into four categories according to the type of algorithm that they use.

2.2.1. Traditional approaches

In this scenario, there are two ways of seeing a missing edge: the first one, and the more obvious, is simply assuming that the graph is incomplete and it needs to be filled. The second one, instead, assumes that the network is evolving over time and we need to predict the new links for the next time stamps. Traditional approaches focused on the second definition of the problem [34]. The traditional *topology-based* approach has as a key assumption the fact that if two nodes are similar enough, for example, if they have enough neighbours in common, sooner or later they will be connected [32, 34]. A concrete example would be about two people that do not know each other but they have many friends in common, it is easy to think that sooner or later they will become friends too. In this case, our task is called *Neighbour Overlap Detection* [17]. However, there are different ways to express the likelihood to be connected based on the neighbourhood, and there are also different ways of defining the neighbourhood. Usually, the *node degree* is taken into account for these metrics: d_x is the number of nodes connected to the node x.

The simpler *local neighbour overlap* technique is the number of common neighbours. However, since high-degree nodes tend to be connected to more nodes, different ways of coping with this bias have been developed. Sørensen index [51] divides the number of common neighbours by the sum of the degree of the two nodes. Salton index [46], instead, normalizes the number of common neighbours by using the squared root of the product of the nodes' degrees. Jaccard index [21] minimized the effect due to node degree by normalizing by the size of the union of the two neighbourhoods, but without duplicates. Resource Allocation index [69], and Adamic-Adar index [1] use a different approach since they take into account the importance of each common neighbour. Local neighbour overlap techniques are reported in Table 2.1.

The main *global neighbour overlap techniques*, which can be seen in Table 2.2, are Katz index [25], which counts the path of all lengths between two nodes, it uses a parameter β to change the importance of long and short paths, and LHN index [30] that can be seen as a normalized version of Katz [25] index since it is not biased on node degree. In the formula, m is the total number of edges in the graph, λ_1 is the largest eigenvalue of the identity matrix A, I is a $|V| \times |V|$ identity matrix indexed in a consistent manner with A.

Table 2.1: Local Neighbour Overlap techniques.

Name	Formula
Number of common neighbours [32]	$S(x,y) = \lvert N(x) \cap N(y) \rvert$
Sørensen index [51]	$S(x,y) = \frac{2\lvert N(x) \cap N(y) \rvert}{d_x + d_y}$
Salton index [46]	$S(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\sqrt{d_x \times d_y}}$
Jaccard index [21]	$S(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\lvert N(x) \cup N(y) \rvert}$
Resource Allocation index [69]	$S(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{d_u}$
Adamic-Adar index [1]	$S(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(d_u)}$

Table 2.2: Global Neighbour Overlap techniques.

Name	Formula
Katz index [25]	$S(x,y) = \sum_{i=1}^{\infty} \beta^i A^i[x,y]$
LHN index [30]	$S(x,y) = \mathbf{I}[x,y] + \frac{2m}{d_x d_y} \sum_{i=0}^{\infty} \beta^i \lambda_1^{i-1} \mathbf{A}^i[x,y]$

2.2.2. Logical Rules

Knowledge graph completion has been addressed also by rule-based methods. A noteworthy approach is AnyBURL [35]. The proposed model is a bottom-up technique developed to efficiently learn logical rules from large knowledge graphs. The rules learnt are in the form of Equation 2.1:

$$h(c_0, c_1) \leftarrow b_1(c_1, c_2), ..., b_n(c_n, c_{n+1}) \tag{2.1}$$

where on the left side there is the *head* of the rule and on the right side there is the *body* of the rule. The rule previously showed is an "if-then" rule in the form of implication as shown in Equation 2.2:

$$X \rightarrow Y \text{ (if X then Y)} \tag{2.2}$$

The learning algorithm is shown in Figure 2.3. Starting from $n = 2$, the algorithm learns rules of length $n - 1$. Then n is increased when the ratio between the number of rules found in the current iteration that were also found in previous iterations and the total number of rules found in the current iteration is above a certain threshold *sat* such as

AnyBURL($\mathbb{G}, s, sat, Q, ts$)

$n = 2$
$R = \emptyset$
loop
  $R_s = \emptyset$
  $start = currentTime()$
  repeat
    $p = samplePath(\mathbb{G}, n)$
    $R_p = generateRules(p)$
    for $r \in R_p$ **do**
      $score(r, s)$
      if $Q(r)$ **then**
        $R_s = R_s \cup \{r\}$
      end if
    end for
  until $currentTime() > start + ts$
  $R'_s = R_s \cap R$
  if $|R'_s|/|R_s| > sat$ **then**
    $n = n + 1$
  end if
  $R = R_s \cup R$
end loop
return R

Figure 2.3: AnyBURL rules generation [35].

99%. The algorithm stops after a previously established time ts, a good value can be 100 seconds. Rules are stored if they are above a certain threshold confidence Q that can be also very low like 0.0001. At the prediction phase, triplets are ranked according to the confidence of the rules that support them. If there are ties, the comparison is performed between the rules with the second largest confidence that support the triplets and so on, until there is a winner. The authors claim that this method achieves state-of-the-art performance despite not being based on ML and having a short time to train.

2.2.3. Machine Learning

ML focused on learning a low dimensional representation of a node, the *embedding*. These methods usually adopt an *encoder-decoder* architecture using shallow encoding [17]. In this case, the *encoder* is simply a lookup table that maps a node to a vector. The *decoder*, instead, uses the embedding to reconstruct some graph statistics. In the use case of link prediction, the decoder may want to reconstruct some close nodes that could be candidates for a connection. The objective of the training is to find an embedding that is able to

give the right output when the decoding is applied.

ML multi-relational decoders can be characterized by their capability to represent different logical patterns on relations [17]. Considering nodes h, and t, a relation r, and E the set of facts in the form h, r, t, we can have four types of relations:

- Symmetric relation: $(h, r, t) \in E \iff (t, r, h) \in E$.
- Anti-symmetric relation: $(h, r, t) \in E \implies (t, r, h) \notin E$.
- Inverse relation: $(h, r, t) \in E \iff (t, \hat{r}, h) \in E$.
- Composed relation: $(h, r_1, t) \in E \wedge (t, r_2, x) \in E \implies (h, r_3, x) \in E$

Different ML models are able to express different logical patterns. Below, some meaningful decoder methods are described.

TransE and early approaches

A model which is imperative to mention is TransE [7] which is one of the first decoders developed. The idea behind this method is that relations are translations in the embedding space. This method learns a low-dimensional vector for nodes and links. When applied for *tail prediction*, after having learnt the embedding h for a *head* node, and the embedding r for a relation, it computes the *tail node* $t = h + r$, as seen in Figure 2.4, and, as result, nodes whose embedding is close to t are candidates for a relationship. This method is *transductive* and therefore cannot be applied to nodes that are not present in the training set, moreover, it cannot model symmetric relations. The number of parameters used is $O(n_e k + n_r k)$, where n_e is the number of entities in the graph, n_r is the number of relations and k is the embedding size. TransE [7] can be considered the pioneering approach to representation learning since it showed how powerful embedding-based approaches could be.

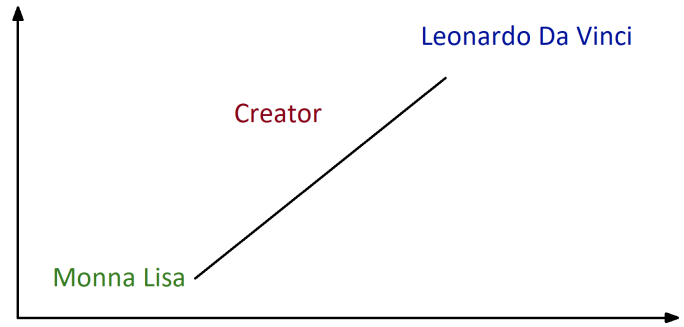


Figure 2.4: Example of a fact modelled by TransE.

Another important model is DistMult [63] which uses an n-hot feature vector to represent embeddings and cosine similarity between $h \cdot r$ and t as scoring function. It outperforms TransE [7], and it is able to use a similar number of parameters since it forces embeddings to be diagonal matrices. The drawback is that it treats all relations as symmetric, therefore it cannot model anti-symmetric, inverse, and composed relations [17].

ComplEx [55] is a tensor factorization model for link prediction that uses complex embeddings to solve the expressivity problems of previous models like DistMult [63]. In particular, the imaginary part of the embedding allows ComplEx to represent also anti-symmetric relations. The only logical pattern that it cannot represent is composition [17]. As a result, it achieves better performance at link prediction when compared with TransE [7], and DistMult [63].

RotatE

RotatE [52] is a knowledge graph embedding method which represents nodes as complex vectors and edges as rotations in complex vector space. It focuses on *tail prediction*. More formally, the model structure is shown in Equation 2.3 and represented in Figure 2.5:

$$t = h \circ r \tag{2.3}$$

where $h, t, r \in \mathbb{C}^k$ are the embedding, respectively of the *head* node, the *tail* node, and of the relation, while $\circ$ is the Hadamard (element-wise) product, and k is the size of the embedding space. If compared with TransE [7], RotatE is also able to model antisymmetry, inversion, and composition, but also symmetric relations. This is possible by using an arbitrary vector r, that satisfies $r_i = \pm 1$. The space and time complexity are linear in the size of the embedding of entities and relations [29, 67]. It is considered the baseline model for transductive link prediction [14].

QuatE

QuatE [67] embeddings lie in the quaternion hypercomplex plane. As a result, relations are quaternion rotations that have two planes of rotations. While the concept of geometric rotation is similar to the one of RotatE [52], QuatE is a semantic matching model, while RotatE is a translational model, however, it also focuses on *tail prediction*. It can be seen as a generalisation of ComplEx [55], and DistMult [63]. Quaternions allow avoiding the gimbal lock (loss of one degree of freedom) and are more stable and efficient than rotation matrices. QuatE method has two steps. First, the head quaternion is rotated using the

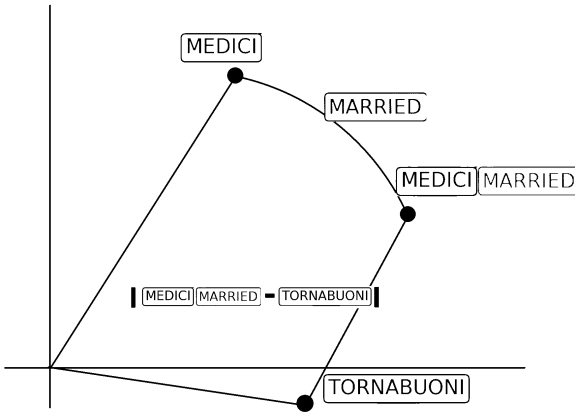


Figure 2.5: RotatE models r as rotation in complex plane, adapted from [52], using Florentine families from the 15 $^{\text{th}}$ century [42] data.

unit relation quaternion as can be seen in Equation 2.4:

$$Q'_h = Q_h \otimes W^<_r \tag{2.4}$$

where $\otimes$ is the Hamilton product, Q_h is the quaternion embedding of the head calculated as in Equation 2.5:

$$Q_h = \{a_h + b_h\mathbf{i} + c_h\mathbf{j} + d_h\mathbf{k} : a_h, b_h, c_h, d_h \in \mathbb{R}^k\} \tag{2.5}$$

with k that is the size of the embedding vector, and $W^<_r$ is the normalized quaternion of the relation calculated as in Equation 2.6:

$$W^<_r = \frac{W_r}{|W_r|} = \frac{a_r + b_r\mathbf{i} + c_r\mathbf{j} + d_r\mathbf{k}}{\sqrt{a_r^2 + b_r^2 + c_r^2 + d_r^2}} \tag{2.6}$$

The second step consists in tackling the link prediction problem as a classification task. The scoring function is shown in Equation 2.7:

$$\phi(h, r, t) = Q'_h \cdot Q_t \ = \langle a'_h, a_t\rangle + \langle b'_h, b_t\rangle + \langle c'_h, c_t\rangle + \langle d'_h, d_t\rangle \tag{2.7}$$

which is the quaternion inner dot product between the embedding of the tail and the rotated embedding of the head obtained as explained before in Equation 2.4. During the

training process, the head relation is rotated to maximize or minimize the dot product with the tail, depending on the existence of the relation in the dataset. Its time complexity is linear in the size of the embedding. QuatE can represent symmetric, antisymmetric and inversion relations, but it cannot represent composition. However, the authors argue that it is intended since multiple composition patterns might exist in the same knowledge graph. QuatE achieves state-of-the-art performance if compared with previous shallow embedding models.

TuckER

This method [4] is based on Tucker Decomposition [56]. A tensor can be decomposed into a smaller tensor and a set of matrices as shown in Equation 2.8:

$$X \approx Z \times_1 A \times_2 B \times_3 C \tag{2.8}$$

having $Z \in \mathbb{R}^{P \times Q \times R}, X \in \mathbb{R}^{I \times J \times K}, A \in \mathbb{R}^{I \times P}, B \in \mathbb{R}^{J \times Q},$ and $C \in \mathbb{R}^{K \times R}$. The idea is to have the same embedding matrix E for object and subject identity as shown in Equation 2.9:

$$E = A = C \in \mathbb{R}^{n_e \times d_e} \tag{2.9}$$

and having a relation embedding matrix as shown in Equation 2.10:

$$R = B \in \mathbb{R}^{n_r \times d_r} \tag{2.10}$$

where n_e is the number of entities, n_r is the number of relations, d_e is the dimensionality of the embedding of the entities, and d_r is the dimensionality of the embedding of the relations. The probability of a triple being true is calculated as in Equation 2.11:

$$S(\phi(h, r, t)) \tag{2.11}$$

where S is the logistic sigmoid function: $S(x) = \frac{1}{1+e^{-x}}$, while the scoring function is shown in Equation 2.12:

$$\phi(h, r, t) = W \times_1 e_h \times_2 r_r \times_3 e_t \tag{2.12}$$

with $e_h, r_r,$ and e_t that are the rows of the entity matrix of the entities h and t and the row of the relation matrix of the relation r. While $W \in \mathbb{R}^{d_e \times d_r \times d_e}$ is the core tensor. TuckER is fully expressive and the number of parameters is linear in the size of the dimensionality of the entities and relations embeddings. At link prediction, it outperforms some shallow embedding models, including DistMult [63], ComplEx [55], RotatE [52], TransE [7], but

also neural network models, including ConvE [9], and R-GCN [48].

2.2.4. Neural Networks

Recent years have been a golden age for Deep Learning. This area showed superior performance compared with traditional Machine Learning models in different fields, including computer vision and natural language processing [61].

There are various types of GNNs, and to distinguish them in this work, the taxonomy of Wu et al. [61] will be followed. The first GNNs developed are called *Recurrent Graph Neural Networks (RecGNNs)* [61]. The Graph Neural Network model (GNN) [47] has been the early approach in this area. This method is a generalisation of previously developed Recursive Neural Networks [12, 47]. The idea is to attach a *state* to each node, consisting of neighbourhood information, like the other connected nodes.

Another type of GNNs are called *Convolutional Graph Neural Networks (ConvGNNs)* [61]. They are further divided into two categories: *spectral-based* and *spatial-based* [61].

Spectral-based approaches define the convolutional operation from the signal processing perspective, in the sense that the convolutional operation can be interpreted as denoising graph signals [61]. Spectral-based methods assume a fixed graph and generalize poorly on new graphs [61]. One noteworthy model is Graph Convolutional Network (GCN) [28]. It uses convolutional layers to approximate localized spectral filters on graphs. The convolutional operation scales better on large graphs with high node degree distribution if compared to GNNs.

Spatial-based ConvGNNs inherit ideas from RecGNNs to define graph convolution by information propagation [61]. Basically, the idea is to neighbourhood information using a message-passing algorithm. These methods are usually *inductive*, which means that are able to generalize well on unseen nodes. A famous model is GraphSAGE [18]. Basically, the idea is to train a set of aggregator functions that learn to aggregate feature information from a node's local neighbourhood. As the name suggests (SAmple and aggreGatE), the node embedding is created in two steps. The first step is to sample a fixed size of neighbours that are connected to the current node representation. Then an aggregation function is applied to the current node representation h_v^{n-1} and the previously collected representations of the nodes in the neighbourhood. After the aggregation function, a non-linearity σ is applied through a fully connected layer, and the result is the new node representation h_v^n. The algorithm can be seen in Figure 2.6. The idea is that by stacking more layers, the embedding of a node will have more and more information on a larger neighbourhood. GraphSAGE [18] proposes different types of aggregation functions,

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 2.6: GraphSAGE embedding generation [18].

including Mean aggregator, LSTM aggregator and Max-Pooling aggregator. This model can be seen as a continuous approximation of the Wesfeiler-Lehman (WL) Isomorphism Test [8, 18]. Research continued in this direction, and it pointed out that GNNs like GraphSage [18] have limited discriminative power [62]. Graph Isomorphism Network (GIN) [62], and later Identity Aware GNNs (ID-GNN) [65] were proposed to solve this problem, however such innovative GNNs are out of scope since they cannot be applied to KGs.

2.2.5. Application of Neural Networks to Knowledge Graphs

Neural networks described so far cannot be applied directly to Knowledge Graphs since they use node features to create embeddings and they do not take into account relation types. However, research adapted some of those models also for this use case. Here I mention the state-of-the-art NNs-based models that can be applied to KGs.

Naïve approaches to Knowledge Graphs: R-GCN

One of the first ConvGNNs applied to Knowledge Graphs is R-GCN [48]. This model is an extension of GCN [28]. The idea is to create an autoencoder that uses as encoder a ConvGNNs to produce latent feature representations of entities and a tensor factorization model as a decoder to be used for link prediction. The authors used as decoder DistMult [63]. GCN [28] can be also seen as a special case of a differentiable message-passing framework whose propagation model can be expressed as in Equation 2.13:

$$h_i^{l+1} = \sigma(\sum_{m \in \mathbb{M}_i} g_m(h_i^l, h_j^l)) \tag{2.13}$$

where $h_i^l \in \mathbb{R}^{d^{(l)}}$ is the hidden state of the node v_i in the l-th layer of the Neural Network (NN), while $d^{(l)}$ is the dimensionality of this layer representation, $\sigma(\cdot)$ is an element-wise activation function like ReLU, and g_m is a message aggregation function that can be a neural network like function or a linear transformation in the form of Equation 2.14:

$$g_m(h_i, h_j) = Wh_j \tag{2.14}$$

Therefore, the proposed R-GCN adapts the message-passing framework showed in Equation 2.13 as described in Equation 2.15:

$$h_i^{l+1} = \sigma(\sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}) \tag{2.15}$$

where N_i^r is the set of neighbour indices of node i under relation $r \in R$, $c_{i,r}$ is a problem-specific normalization constant that can be learned or chosen a priori, e.g. $c_{i,r} = |N_i^r|$. The idea is to accumulate transformed feature vectors of neighbouring nodes through a normalized sum. At the end of each layer, each node has a special self-connection to be informed of the previous representation updates. A representation of a layer can be seen in Figure 2.7. This architecture allows stacking multiple layers to model dependencies between different relational steps. The input of the first layer can be a one-hot encoding for each node of the graph if no other features are present. This approach has problems since there is a large growth in the number of parameters when there are many relation types. Such a problem has been addressed by applying *basis* and *block-diagonal* decomposition to the weight matrices [28]. On link prediction, an R-GCN encoder with a DistMult [63] decoder outperforms machine learning models such as TransE [7], and ComplEx [55].

ConvE

ConvE [9] is a CNN that can create deep node embeddings and can be used for Knowledge Graph Link prediction. Its inference process is shown in Figure 2.8. In particular, this model reshapes and concatenates entity and relation embeddings in order to create an "image". In this way, 2D Convolutional layers can be applied. The resulting feature map tensor is then projected into a fully connected layer of embedding dimension k. Then after a Matrix multiplication with an entity matrix, a Logistic sigmoid is applied to perform the prediction. Formally, the architecture's scoring function is shown in Equation 2.16:

$$\psi_r(e_s, e_o) = f(vec(f([\bar{e_s}; \bar{r_r}] * \omega))W)e_o \tag{2.16}$$

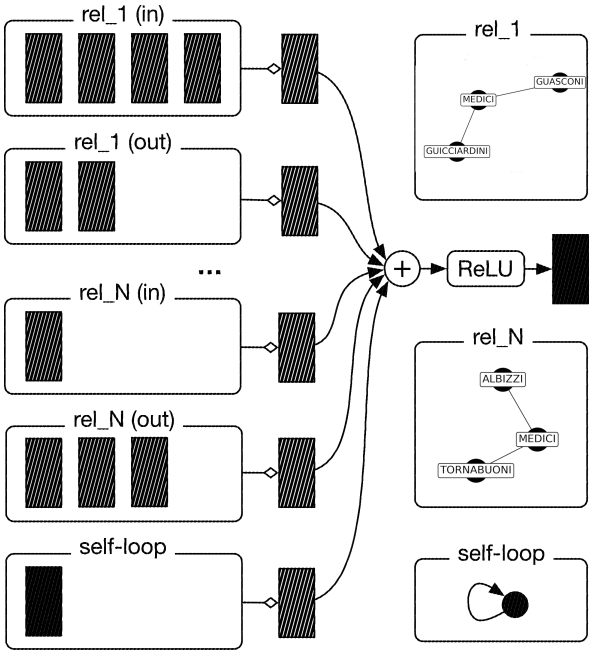


Figure 2.7: A single R-GCN layer, adapted from [48], using Florentine families from the 15 $^{\text{th}}$ century [42] data.

where $r_r \in \mathbb{R}^k$ is a relation parameter depending on r, $\bar{e_s}$ and $\bar{r_r}$ are the 2D reshaping of e_s and r_r, while e_s, and e_o are the embedding of the entities s and o, and the non-linearity f used is the ReLU. Then after the score, the logistic sigmoid function is applied and the probabilities of links are obtained as in Equation 2.17:

$$p = \sigma(\psi_r(e_s, e_o)) \tag{2.17}$$

Moreover, Batch Normalization is used after each layer to increase the convergence rate, and dropout is used for regularization. It is applied on the embeddings, on the feature maps obtained by the convolution, and also on the fully connected layer. ConvE achieves better or similar performance than previous models like R-GCN [48], DistMult [63], and, ComplEx [55] while using fewer parameters.

Modern approaches to Knowledge Graphs: NodePiece

Inspired from Natural Language Processing (NLP), NodePiece [14] is an encoder that uses tokenization to reduce parameter complexity, increase generalization capability, and represent new entities using the learnt vocabulary. The idea is to have a set of *atoms* made by *anchor* nodes and all relation types to construct the embedding of a node. NodePiece tokenization strategy is represented in Figure 2.9, and works as follows: a target node

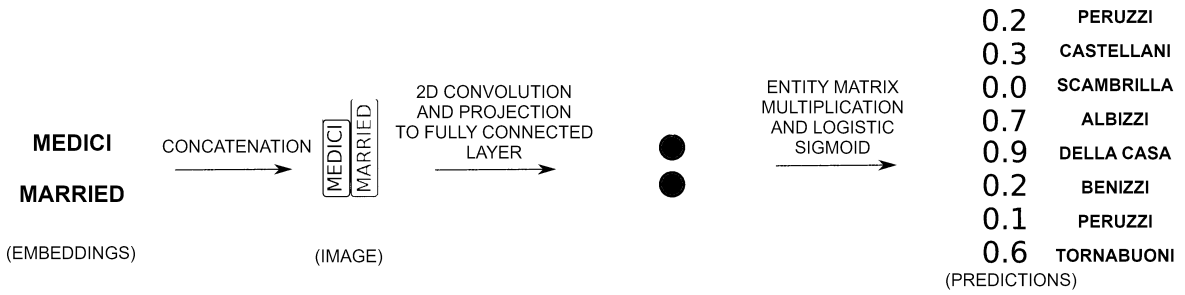


Figure 2.8: ConvE inference process, adapted from Dettmers et al. [9], using Florentine families from the 15 $^{\text{th}}$ century [42] data.

is tokenized into a hash of the k anchors, which can be the k closest one or k anchors chosen using a random strategy, their shortest path distance from the target node, and the outgoing relations form the target node. This means that if a node has an incoming edge, it will be modelled using the inverse relation. Formally, the embedding is obtained as showed in Equation 2.18:

$$hash(n) = [\{a_i\}^k, \{z_{a_i}^k\}, \{r_j\}^m] = [a_n + z_{a_n}, r_n] = [\hat{a_n}, r_n] \in \mathbb{R}^{(k+m)\times d} \tag{2.18}$$

with m that is the size of the relational context, a_n and r_n that are taken from the vocabulary, and anchor distance z_{a_n} from $Z \in \mathbb{R}^{(diameter(G)+1)\times d}$. d, instead, is the size of the embedding vector. Then an encoding function is applied to obtain the embedded vector as in Equation 2.19:

$$enc = \mathbb{R}^{(k+m)\times d} \rightarrow \mathbb{R}^d \tag{2.19}$$

The vocabulary is made by anchor nodes and relation types as described in Equation 2.20:

$$V \in \mathbb{R}^{|V|\times d} = A + R \tag{2.20}$$

while the anchor nodes are a subset of the total nodes $A \subset N$, the relation types are the entire set of relations R since usually the total number of relation types is much lower than the total number of nodes $|R| \ll |N|$. The dencoding function can be a Multi Layer Perceptron (MLP) [53] or a Transformer that uses the attention mechanism [57]. NodePiece achieves very good performance on inductive link prediction.

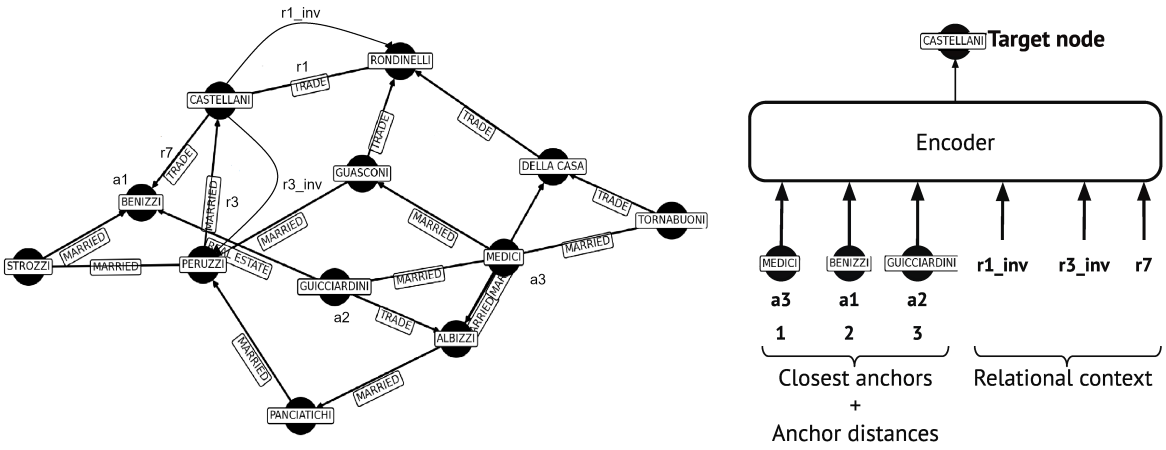


Figure 2.9: NodePiece tokenization strategy, adapted from [14], using Florentine families from the 15 th century [42] data.

2.3. Microservices

Modern web applications are getting bigger and bigger. As a result, they are not easy to maintain. To solve this problem, applications have been split into smaller applications that can be deployed, scaled, and tested independently and have a single responsibility. These small applications are called microservices [54] where each service performs a specific function and communicates with other services through APIs.

Microservices have been widely adopted as architecture by many tech giants including Microsoft, Google, Amazon [38], and Netflix [54]. Since each service is independent and can be developed and deployed separately, it allows to speed up development cycles and enables teams to respond to changes more quickly. However, for the same reason, there is an additional layer of complexity. In particular, when considering large-scale applications, it is not easy to understand how the microservice architecture will behave a priori. Therefore tracing tools have been developed to analyse the application's structure. Famous examples are: Dapper [50], Jaeger [23], Zipkin [41], and OpenTelemetry [40]. The aim of the tracing operation is to build a microservice tracing graph. An example can be seen in Figure 2.10. Useful applications are improving performance by bottleneck/critical path detection, request time measurement, root cause analysis, and automatic scaling.

2.3.1. Tracing Tools

A microservice tracing graph is used to track the user request path inside the system. Nevertheless, messages exchanged between microservices require overheads in order to

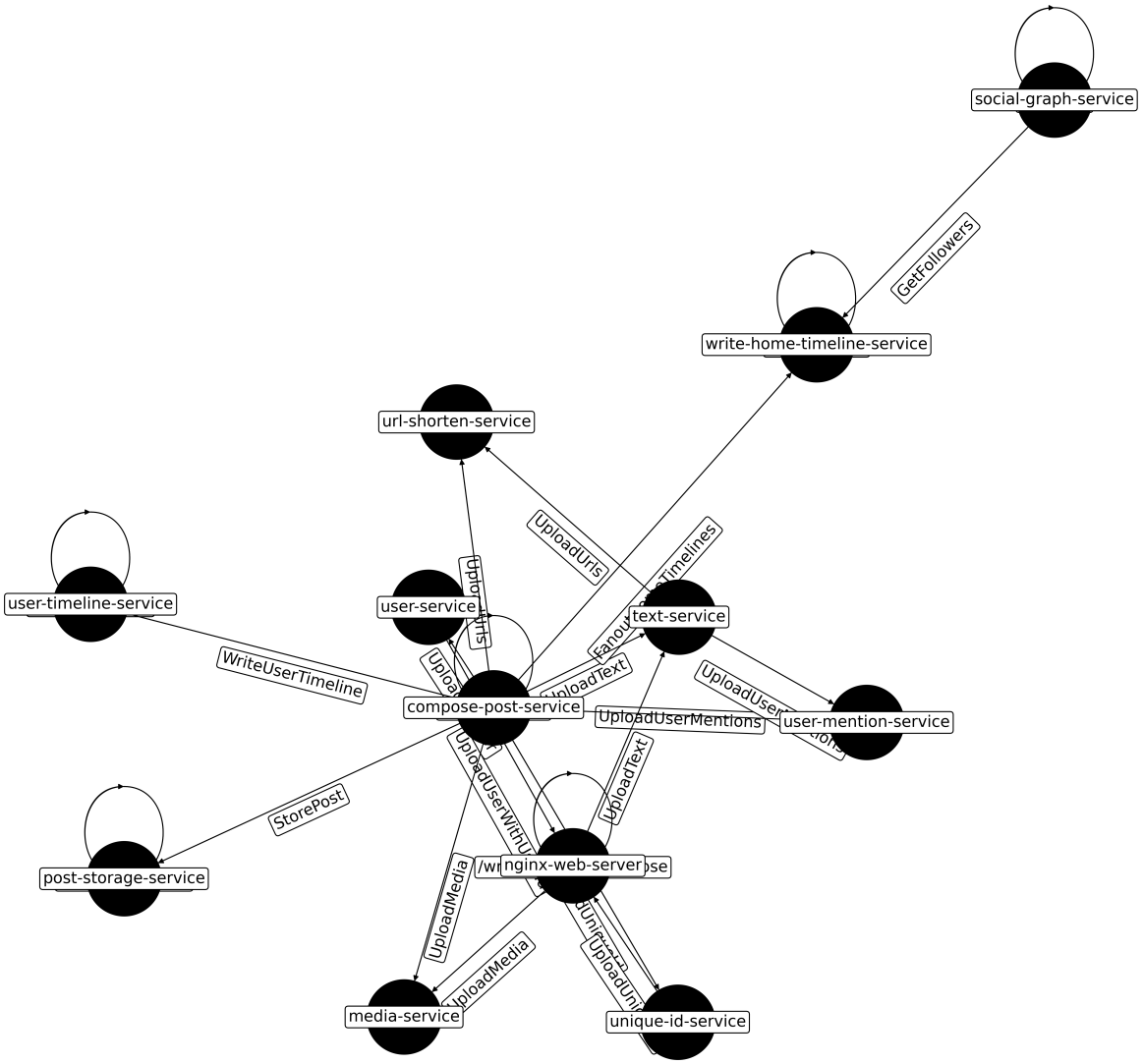


Figure 2.10: Example of a microservice graph sampled from https://gitlab.mpi-sws.org/cld/trace-datasets/deathstarbench_traces (drawn using newtworkx).

create these graphs. In particular, the requests are monitored by attaching relevant contextual metadata along with them during their execution [49]. The overhead results in performance degradation. Therefore, tracking many requests on massively distributed applications seems unreasonable, leading to the creation of incomplete graphs. Often sampling is used to reduce the overhead, however, when this technique is applied, obtained traces are less representative [50].

There are two ways to perform sampling, namely *head*-based, and *tail*-based sampling [49]. When using *head*-based sampling, as it happens for Dapper [50], and Jaeger [23],

the decision of collecting a trace or not is performed when the front-end web service receives a request from the user. This means that a trace is sampled as a whole or not sampled at all. Therefore it is possible to reduce the overhead, by simply reducing the probability of a request being tracked. *Tail*-based sampling, instead, registers the path of each trace, and when they are complete it chooses if storing them or not. It can be useful to track specific cases such as traces with errors, or traces that exceed a certain threshold latency. Nevertheless, it results in a higher overhead than *head*-based sampling. *Tail*-based sampling is supported by OpenTelemetry [40].

As a result, it is not possible to have a low overhead and high representative samples at the same time. For this reason, this project explores the use of link prediction tools to solve this problem.

3 | Method

In this chapter, I describe the research methodology and methods that are used in the degree project.

3.1. Comparative Analysis

The main task of the degree project consists of a comparison of different methods for link prediction. The models compared are RotatE [52], QuatE [67], TuckER [4], ConvE [9], R-GCN [48], NodePiece [14], and AnyBURL [35]. The metrics used are *Mean Reciprocal Rank (MRR)*, *Hits@K (H@K)* with k equals to 1, 3, 5, and 10, in the *filtered* scenario, and using *max*, *average*, and *min* policies, and also *training time*, *inference time per trace*, and *memory usage*. In the case of AnyBURL [35], since its custom implementation, it will be evaluated only on *both* prediction, using *min* policy, and will be visualized only in the averaged *both* prediction case.

A more detailed explanation will be performed in the following sections.

3.2. Choice of the Research Method

For this thesis, following the framework of Håkansson [16], it has been decided to perform a quantitative research with a deductive approach and an experimental strategy.

This choice is supported by the fact that previous studies that focused on comparing link prediction methods, including Rossi et al. [29] followed the same methodology. Moreover, a quantitative analysis is a standard in the ML area. Additionally, by adopting a quantitative research approach, the thesis aims to provide objective and measurable results. This allows for a systematic evaluation of different link prediction methods, enabling comparisons and drawing reliable conclusions. Furthermore, the chosen methodology ensures that the experiment can be replicated by other researchers, enhancing the transparency and credibility of the findings. Lastly, the framework of the thesis lies in the quantitative area, since data is collected through experiments.

However, while comparing the performance of the specific metrics has been performed objectively by looking at the best value, it has not been the case when considering the overall performance. In particular, in a subjective manner, a model has been considered to be suitable to be the best if its performance on the additional requirements of training time, memory usage, and inference time were at least average and better than the threshold if present. Then the models have been compared on the prediction accuracy. It seems a good trade-off because although the main quality of a model is the reliability of its predictions, it is necessary for an application to fulfil some service level agreement and therefore a minimum value of performance on the additional requirements.

Moreover, in the overall comparison, a qualitative comparison is necessary, otherwise, it would be needed to perform a quantitative comparison on many datasets, and it is out of the scope of this thesis due to the limited time available.

3.3. Metrics

In the prediction phase, when we have an incomplete triple $\langle h, r, ? \rangle$ and we want to predict the missing tail, or vice versa when we want to predict a missing head, we chose the node that results in the highest score computed as in Equation 3.1:

$$t = \underset{n \in N}{argmax}\ \phi(n, r, t) \text{ where N is the set of nodes} \tag{3.1}$$

To do so, we compute the score for each node and relation, and based on the relative score, we associate a *rank* Q to each of them.

In this thesis, to assess the accuracy of the prediction the framework described by Rossi et al. [29] will be followed. The two metrics presented are *MRR* and *H@K*.

- MRR $= \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{q} \in [0, 1]$, the higher the better. It is the average of the inverse ranks.
- H@K $= \frac{|\{q \in Q\ :\ q \leq K\}|}{|Q|} \in [0, 1]$, the higher the better. It is the fraction of prediction ranks that are equal to or below a certain threshold K. Common values for K are 1, 3, 5, and 10.

Moreover, there are different settings in which the rank can be computed:

- *Raw Scenario*: valid entities outscoring the target are considered mistakes, and they do contribute to the ranking.
- *Filtered Scenario*: valid entities outscoring the target are not considered mistakes

and they do not contribute to the ranking.

Furthermore, there are different *tie-breaking policies* that are used when multiple entities have the same score as the target one:

- *min*: the target is given the lowest, and therefore best, rank among all entities. It may result in artificially boosting the benchmark.
- *average*: the average rank is assigned to the target.
- *random*: a random rank chosen among the tied entities is assigned to the target, with a large number of measurements it is equivalent to *average*.
- *max*: the highest, and therefore worst, rank among all entities is given to the target.

Metrics can be computed for *head* prediction or *tail* prediction only, but their results can be averaged to obtain the *both* prediction score.

The previously mentioned metrics will be used to compute the accuracy of the models that will be compared in this thesis. To have a more detailed comparison, both MRR and H@K will be used. Instead, only *Filtered* scenario will be taken into consideration due to software limitations. *Max*, *average*, and *min* will be used as *tie-breaking policies* when allowed by the software, in order to have a complete overview of the performance of the models.

3.4. Model Selection Rationale

As reported in Chapter 2, many models have been developed for link prediction, nevertheless, as previously stated, the comparison will be performed between RotatE [52], QuatE [67], TuckER [4], ConvE [9], R-GCN [48], NodePiece [14], and AnyBURL [35].

The rationale behind this choice is that the comparison should be performed between state-of-the-art models that claim to fulfil the requirements, and it should include representatives of different approaches. However, it is important to say that the comparison is limited to models implemented in modern Python packages since the goal of this thesis is to compare the methods and not to develop new ones.

- RotatE [52] has been chosen since it is considered a transductive link prediction baseline [14].
- QuatE [67] has been chosen since it is a new method among ML models, it is based on hypercomplex embedding which is unprecedented in this area and still achieves linear inference time in the embedding size.

- TuckER [4] has been chosen since it is a tensor decomposition model based on Tucker factorization [56] and for its optimal performance.
- ConvE [9] has been chosen as representative of Convolutional Neural Networks (CNNs).
- R-GCN [48] has been chosen as representative of GNNs.
- NodePiece [14] has been chosen for its peculiar anchor-based approach, moreover, it will be used as the baseline for inductive link prediction for its inherent inductive nature.
- Anyburl [35] has been chosen as representative of rule-based methods and for its fast training time.

Other models have been the subject of a preliminary literature study, but have not been selected for comparison since they did not fulfil the criteria. In particular, ConvKB [37] has not been selected since it is a CNN based method but it is outperformed by ConvE [9]. ComplEx [55] despite its good reputation has not been selected since it is generalised by QuatE [67], and TuckER [4]. DistMult [63] and HolE [39] are surpassed by many other models developed including QuatE [67], and the others ML models selected for the comparison. Therefore they cannot be considered state-of-the-art models. TransE [7] has not been selected for the comparison, because despite being famous, it is one of the first models developed, and now its performance is not good as newer models one. TransH [60] and TransD [24] are both part of the initial current derived from TransE [7] and are not state-of-the-art models. SimplE [26] has not been selected for the comparison, because despite its premises of using few parameters, it is actually outperformed on this field by TuckER [4].
Moreover, all topology-based methods described in Section 2.2.1 will be used for the comparison, despite they are not expected to lead to good results.

3.5. Additional Metrics and Requirements

As previously stated, an important part of this thesis project is comparing the models on metrics other than accuracy, since they have often been neglected from research. In particular, both *training time* and *inference time* will be measured. Moreover, the inference time per triplet could be considered a proxy for the computational cost. The *memory usage* instead will be measured through the size of the parameters. These requirements have been identified according to open problems pointed out from previous studies [19, 38, 45]. The additional requirements have been monitored in the same way regardless of

the modelling case.

Testing time per trace has been monitored in the traditional prediction approach since having more samples would lead to a more robust result.

Memory usage has been measured just once when all the models have been instantiated for the first time. Since it is measured through the size of the parameters it does not change depending on the approach used or the size of the train set. Moreover, also the number of parameters has been monitored, to understand better what the models are learning.

The training time has been measured on the entire dataset and on a subset which accounted for 29% of the total size, to have some understanding of the scalability of the models. The subset training time has been monitored only for the *static* modelling scenario since it was not possible to create a dataset with the same percentage size without breaking up the different traces.

4 | Method Application

This chapter contains what was actually done for the degree project and a practical description of how the method was applied.

4.1. Training and Testing

This section explains how training and testing have been performed. In particular, it describes the training settings including the hyperparameters, and the different scenarios investigated. Chiefly, it describes how the problem has been modelled using a *static*, and *IO* modelling, and how in both cases the prediction scope has been differentiated using the *traditional* and the *trace-based* prediction approaches. Moreover, it illustrates the *transductive*, and *inductive* settings used in the *traditional* prediction scenario.

4.1.1. Training Settings

All models have been trained in the same setting when possible, in particular, the loss function used is Self Adversarial Negative Sampling (NSSA) loss proposed with RotatE [52], and also used by NodePiece [14]. The dimension of the embedding space has been set accordingly to the papers that presented the models [4, 9, 14, 67]. The learning rate has been set in order to obtain a reasonable training process, in particular, the starting value was 0.01 since it was the default value on pykeen. Higher learning rates have been used when lower learning rates were too low to make the models learn, and they remained stuck with the same loss value. The early stopping patience, defined as the number of training epochs without a decrease in validation error before stopping the training, has been chosen arbitrarily. The batch size of ConvE [9] has been chosen arbitrarily. The number of tokens for NodePiece [14] which is the number of nodes used to represent each entity has been chosen according to Galkin et al. [14]. The same reasoning has been followed when choosing the aggregation, which is the decoder used by NodePiece [14] after the generation of the embeddings. However, its exact structure is the default one of Pykeen. All the other settings are the default ones of pykeen. The RAM usage of AnyBURL [35] has been set to 1 GB since it was the smallest value possible, and the

training time has been set to 100 seconds since according to its paper it was enough time to obtain good results, and using more time would have lead to the generation of more rule and therefore would have increased also the inference time [35]. Thus, having a graph not too large, 100 seconds of training time is a good choice. The training settings are shown in Table 4.1.

Table 4.1: Training setting

Hyperparameter	**Value**	**Affected Models**
Loss Function	NSSA [52]	All models except for AnyBURL [35]
Embedding space dimension	200	All models except for AnyBURL [35]
Optimizer	Adam [27]	All models except for AnyBURL [35]
Learning rate	0.001 0.01 0.05	ConvE [9], and NodePiece [14] TuckER [4] RotatE [52], QuatE [67], and R-GCN [48]
Batch size	512	ConvE [9]
Number of tokens	20	NodePiece [14]
Aggregation	MLP	NodePiece [14]
Maximum train epochs	2000	All models except for AnyBURL [35]
Early stopping patience	20	All models except for AnyBURL [35]
Training time	100 seconds	AnyBURL [35]
RAM usage	1 GB	AnyBURL [35]
All other values	Pykeen default	All models except for AnyBURL [35]

4.1.2. Building the Graph

An important step has been understanding how to build the Knowledge Graph from the available data. In this thesis two different areas of modelling have been identified, the first one focuses on the temporal information of the relations and the second one focuses on the scope of the prediction. For both cases, two different options have been explored. Moreover, the dataset has been split between training, validation, and testing, using two different settings.

Temporal information of relations

The problem addressed here is whether and how to represent the fact that microservice calls follow a certain temporal order. In particular, two different modelling approaches have been developed to understand if the temporal information is relevant or not for making predictions. Since the problem has been modelled as a knowledge graph, it made sense to focus on the relation types.
Therefore, inspired by Qiu et al. [44], data has been modelled in two differed ways:

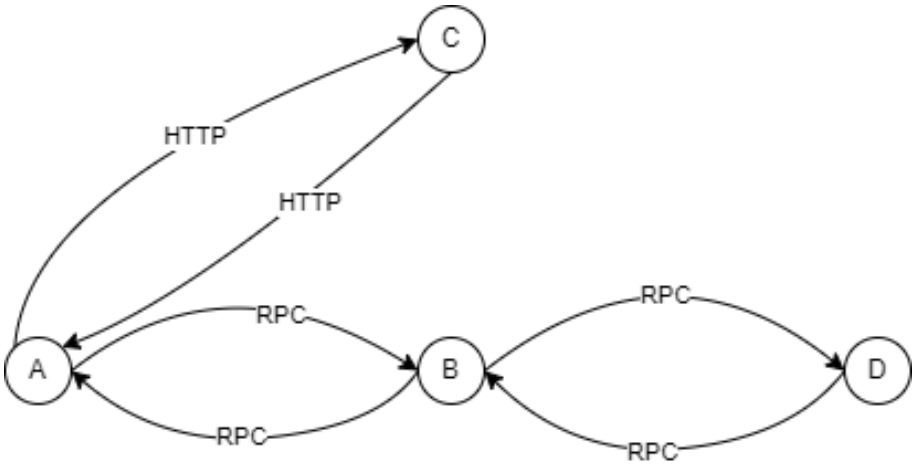
- Static modelling: the temporal information is not taken into consideration at all, nodes are microservices and relations are simply the type of the Remote Procedure Call (RPC) used.
- IO modelling: the number of relation types is increased since for each call, it is specified if the call is an output call (request), or an input call (answer).

A comparison of the two approaches is shown in Figure 4.1.

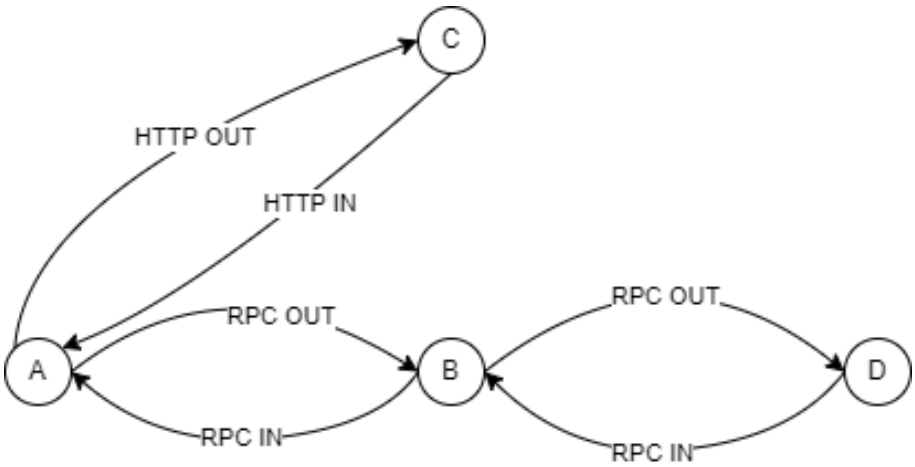
Scope of prediction

Two different approaches have been followed since assessing the performance of a model simulating a real use-case scenario cannot be done by evaluating the results as it is usually done in the literature. Famous KGs used in the literature such as FB15k [7], WN18 [15], and YAGO3-10 [33] have from thousands to ten of thousands of entities, tens of thousands to hundreds of thousands triplets, and a variable number of relations which can go up to few hundred and in some cases even more. Having such data, models are trained on a part of the graph where some links are missing. The missing relations are used in the test set to assess the performance of the models. Therefore methods need to be trained and evaluated on a single graph. However, microservices call graphs are completely different, since they are multiple small graphs corresponding to the different traces. Consequently, it is necessary to develop a different approach to understand how models perform in a real use-case scenario.
Thus, two different approaches have been followed: *traditional* prediction and *trace-based* prediction.

Traditional prediction All the multiple traces have been joined in order to create a single big graph to be used for training and testing, as it is usually done in the literature. Basically, the dataset has been considered as a big graph where it is required to predict the missing links present in the validation and testing set. The case of graph composition when having a dataset with just two traces is represented in Figure 4.2. The validation



(a) Graph modelled using the static approach.



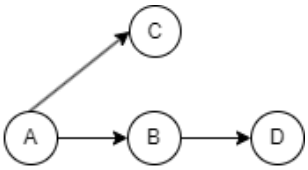
(b) Graph modelled using IO approach.

Figure 4.1: Same data modelled using the two different approaches.

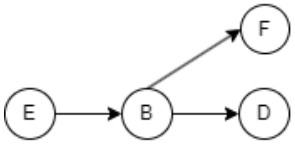
set has been used for the early stopping.

Data splitting for traditional prediction The dataset has been split into train, validation and testing. In the case of traditional prediction, the splitting was performed according to two different settings: *transductive*, and *fully inductive* [13].

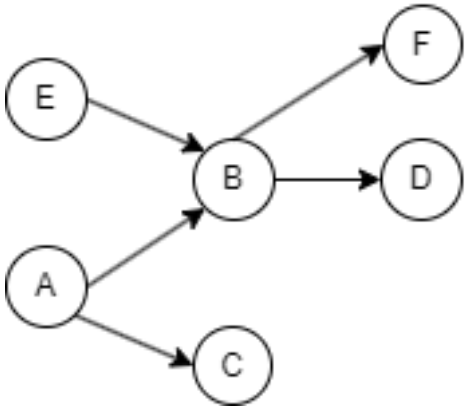
- Transductive setting: the dataset has been split at triplets level in the following way: train set with 64% of the data, validation set with 16% of the data, and test set with 20% of the data. In this case, all the entities and relation types of validation and testing are present also in the training set. An example of the splitting can be seen in Figure 4.3.
- Fully Inductive setting: the entities of the training, validation and test set are not overlapping. As a result, the dataset has been divided at the entity level using the



(a) Example of a trace represented as a tree.



(b) Example of a trace represented as a tree.



(c) Resulting graph. Relations have been omitted to improve readability.

Figure 4.2: Visualization of the creation process of the graph used for training and testing the model according to the traditional prediction approach. A, B, C, D, E, and F are microservices, in figure a, a user submits a request through microservice A and it is then propagated to B, C, and D. Instead in figure b, a user submits a different request through microservice E, and it is then propagated to B, F, and D. Figure c shows the resulting graph obtained by joining the traces on common microservices B, and D.

same proportions of the *transductive* case. However, to guarantee the fully inductive property, 46% of the total triplets have been discarded for the static modelling and 55% for the IO modelling. In the *fully inductive* setting, only NodePiece [14] have been used, since the other models are inherently transductive and cannot generalize on unseen graph nodes. R-GCN [48] should have worked in this scenario, but unfortunately, PyKeen supported only its transductive implementation. An example of the splitting can be seen in Figure 4.4.

Trace-based prediction The models are still trained as it happens for the previous case, but at the prediction phase, their output is restricted to the entities and relations that are known that are belonging to a specific trace. The final performance is then obtained by averaging the single performance of each trace. As will be explained in Section 5.2, there is a certain difference between the structure of the graph and the subgraphs which is attributable to the use case. The reason is that while usually in KGs

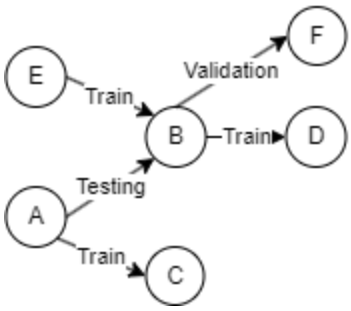


Figure 4.3: Representation of graph seen in Figure 4.2 split in training, validation and testing according to the transductive setting following the traditional approach. Links are labelled according to the dataset to which they belong.

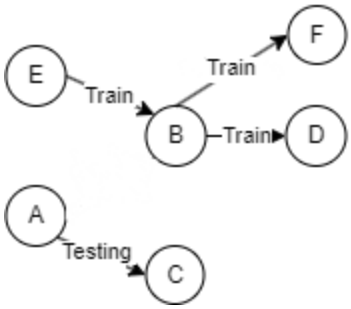


Figure 4.4: Representation of graph seen in Figure 4.2 split in training, and testing according to the fully inductive setting following the traditional approach. Links are labelled according to the dataset to which they belong. One link has been removed to ensure the fully inductive setting, therefore nodes A, and C are never seen by the models during training. The validation set has been omitted for simplicity

all nodes could be linked with each other, it is not the case with microservices calls. In fact, some microservices may not be linked with each other since they are associated with different web services. For example, two functionalities offered by a web app may not share logic at all, and therefore may not have microservices in common. Therefore assessing the performance of a model using the classic approach is not relevant to the use case, especially considering the fact that traces have usually just a few entities and relations.

While the training in approach remains the same as presented in the traditional prediction approach, the testing works as follows:

- For each trace that constitutes the transductive test set, a *train-trace* set and a *test-trace* set are built. The *train-trace* set is the smallest "transductive" set considering only the triplets of the trace. It may or may not be overlapping with the traditional

train set.

- The *test-trace* set is the remaining part that needs to be predicted. It may or may not be overlapping with the traditional train set.
- At the prediction phase, the model output is restricted to consider only the entities and relations present in the *train-trace* set, and it is evaluated the results on the *test-trace* set.

The idea is that the models are tested at trace level in the transductive setting, and the output is reduced to the traces and entities already seen in the training part. Therefore, since the result space is restricted to a single web service, it is not possible to predict a link to a node that is not associated with that trace. For the same reason, this approach cannot be used in the inductive scenario. An example of this approach can be seen in Figure 4.5.

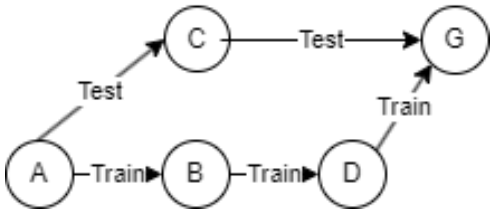


Figure 4.5: Example of a trace split in *train-trace* set and *test-trace* set. Links are labelled according to the set to which they belong.

4.2. Dataset Choice

The dataset used to build the knowledge graphs contains microservice traces since the previously stated requirements are also seen as crucial challenges in microservice-based application [45, 66, 68]. In particular:

- Inference time is considered a critical requirement for applications that make use of microservices graphs, such as CRISP [68] that performs critical path analysis, or DeepTraLog [66] that perform anomaly detection.
- Training time is an essential requirement for the same reasons.
- Memory usage has been considered fundamental by applications that use microservices knowledge graphs to perform root cause analysis [45].

4.3. Platform

Both data exploration and the model comparison have been performed using Python. Data exploration has been performed on my laptop and on Google Cloud Platform (GCP) virtual machines provided by Research Institutes of Sweden (RISE). Instead, the platform used for training and testing the models has been provided by National Academic Infrastructure for Supercomputing in Sweden (NAISS). In the comparison, the computer had as CPU a *Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz* with 32 cores, 100 GB of RAM, and a *Nvidia Tesla T4* as GPU. The ML library used for link prediction was Pykeen. AnyBURL [35], instead, has been developed as a Java application.

5 | Artefacts

In this chapter, the artefacts of the thesis are presented. They include managing and exploring the dataset. Moreover, also some statistics have been computed, in order to have a better understanding of how to model the problem.

5.1. Understanding the dataset

The dataset chosen is Alibaba Cluster Trace v2021 [2]. It contains the detailed runtime metrics of nearly twenty thousand microservices. They are collected from Alibaba production clusters of over ten thousand bare-metal nodes over twelve hours in 2021.

The dataset is made of multiple subgraphs that correspond to different traces. Each trace starts with a user web request to the entering microservice that triggers multiple calls between related microservices.

Each call is between an Upstream Microservice (UM) and a Downstream Microservice (DM). Each entry of the dataset is a call between two microservices. Each call has associated a timestamp, a trace id, an rpc id, the *UM id*, the *DM id*, an *rpc type*, the DM interface id, and the response time of the call. An example of the dataset can be seen in Figure 5.1.

Following the approach explained in Section 4.1, triplets in the form *<head, rel, tail>* have been created in the following way:

- Static modelling: *<DM, rpc type, UM>*;
- IO modelling:
  - *<DM, rpc type in, UM>* if the response time (rt in the table) is positive;
  - *<DM, rpc type out, UM>* if the response time is negative.

An example of the resulting traces for the static modelling can be seen in Figure 5.2. Since a response time lower than 1 ms has been reported as 0 ms, it was not always clear whether a call was an inbound or outbound call, therefore those triplets have been

	traceid	timestamp	rpcid	um	rpctype	dm	interface	rt
26	0b133c19159...	219376	0.1.3.1.4.1...	75e56c8fbb9...	rpc	84f9f68ef00...	59dc80772dd...	-1
53	0b133c19159...	219360	0.1.3.1	20119b0b786...	rpc	4ab265f5451...	95a823f303d...	466
60	0b1339ef159...	175154	0.1.1.2.7.1.17	75e56c8fbb9...	rpc	84f9f68ef00...	59dc80772dd...	-2
64	0b1339ef159...	175155	0.1.1.2.7.1.12	75e56c8fbb9...	rpc	75e56c8fbb9...	0b8b98381d4...	-157
66	0b1339ef159...	175156	0.1.1.2.7.1...	75e56c8fbb9...	rpc	091794afdcf...	7f2e118b038...	1
...	...	...	...	...	...	...	...	...
6088838	0b520694159...	241027	0.1.1.2.7.1...	75e56c8fbb9...	rpc	9a9e8613b6d...	7f2e118b038...	1
6088840	0b520694159...	241025	0.1.1.2.7.1...	75e56c8fbb9...	rpc	9a9e8613b6d...	7f2e118b038...	3
6088841	0b520694159...	241025	0.1.1.2.7.1...	75e56c8fbb9...	rpc	9a9e8613b6d...	7f2e118b038...	1
6088842	0b520694159...	241025	0.1.1.2.7.1...	614c66b178d...	rpc	9a9e8613b6d...	60339798212...	1
6088844	0b520694159...	241022	0.1.1.2.7.1...	175834ec816...	http	9a9e8613b6d...	14e30cd163c...	18

Figure 5.1: Example of Alibaba dataset.

skipped. Other features have not been used since they were not relevant to the problem.

5.2. Data Analysis

Data has been analysed at a higher level, focusing on the number of unique entities, relations, and triplets and on the duration of the response time of each call, but also at the trace level, considering the number of entities and relations per trace. Later, the dataset was processed and prepared to be used properly for training and testing.

5.2.1. Graph level analysis

The dataset was split into several parts and together occupied more than 200 GBs, therefore not a little effort was required to analyse and gather all the data, in fact, the analysis took several hours. Despite this, since not all features were used, and duplicate triplets were removed, since it made no sense to model the problem as a multigraph, the number of total calls has been drastically reduced. In particular, the composition of the final dataset is shown in Table 5.1.

The timing analysis has been performed using the response time of each call and can be seen in Table 5.2. Its results will be used as the baseline for the inference time.

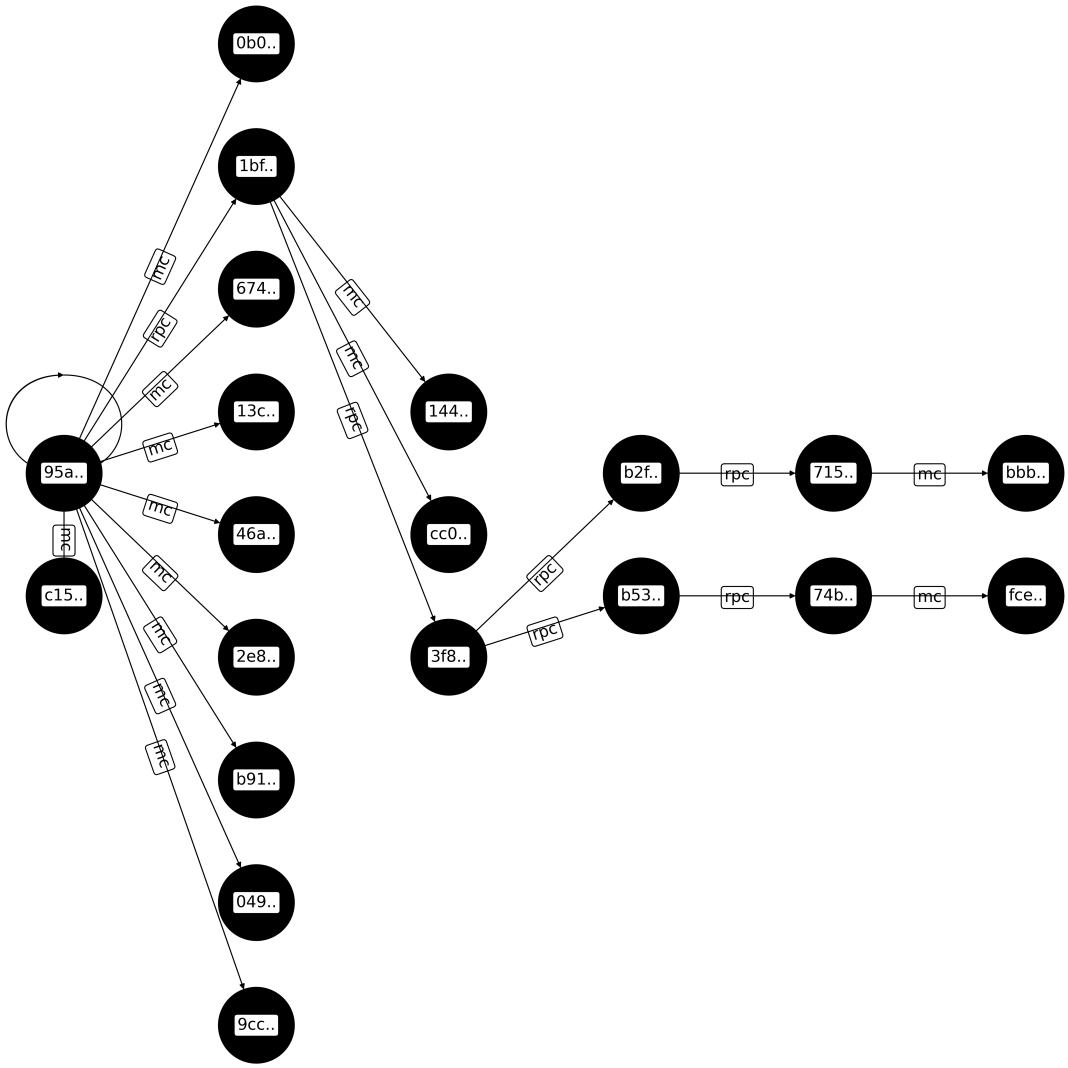


Figure 5.2: Example of an Alibaba trace plotted as a tree.

Table 5.1: Final dataset composition.

Modelling type	Number of unique triplets	Number of unique entities	Number of unique relations
Static	55562	16657	6
IO	64870	16444	8

5.2.2. Trace level analysis

The analysis at the trace level has been performed on a single part of the dataset which accounted for around 0.7% of the entire dataset and around 29% of the total number

Table 5.2: Timing analysis on calls response time.

Mean response time per call (ms)	Standard deviation (ms)	Median response time per call (ms)
10.60	63.34	1.0

of unique triplets and had more than 5 million rows. It took more than 9 hours of computation. The result is shown in Tables 5.3, and 5.4, and visualized in Figure 5.3.

Table 5.3: Analysis on calls at trace level.

Mean number of calls per trace	Standard deviation	Median number of calls per trace
40.48	126.40	10.0

Table 5.4: Analysis on entities at trace level.

Mean number of entities per trace	Standard deviation	Median number of entities per trace
11.42	14.67	7.0

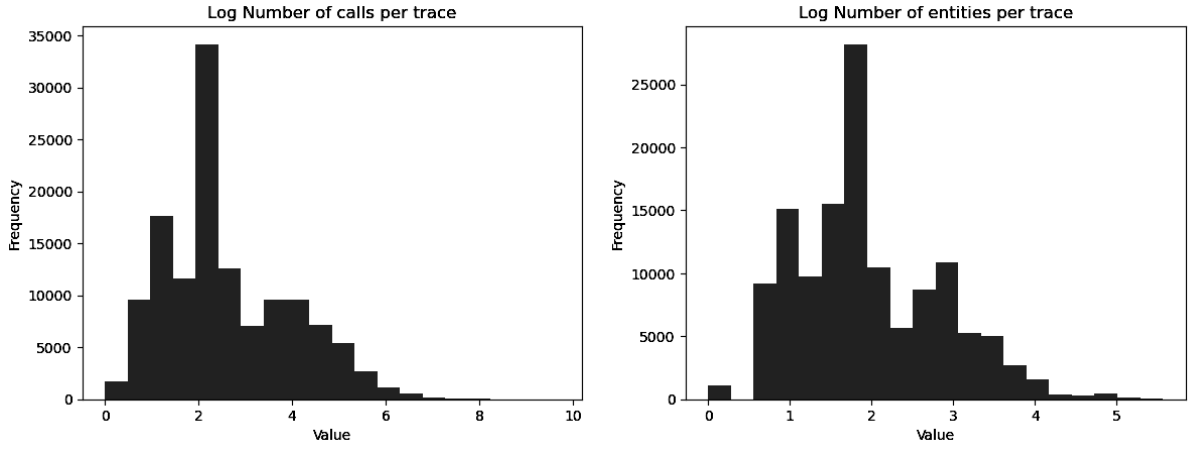


Figure 5.3: Visualization of the analysis on calls at trace level.

While the graph has many triplets and entities, the structure of the subgraphs, the traces, is completely different. The traces have a very low number of calls and touch a low number of entities. These differences are highly related to the use case and will be addressed at the testing phase as explained in Section 4.1.

6 | Experimental Results

In this chapter, the results of the thesis are presented. They include the accuracy performance of the models with *static* modelling, and *IO* modelling, on both *traditional* and *trace-based* prediction scenarios that have been previously described in Section 4.1. Moreover, the performance of the additional training time requirements, inference time per triplet and memory usage will be shown.

The interpretation and discussion of the results will be presented in Chapter 7.

6.1. Case 1: Static Modelling

In this section, the results of static modelling are reported. They include the *traditional*, and *trace-based* prediction approach.

6.1.1. Approach 1.1: Traditional prediction

Here are presented the results divided on the type of prediction: *head*, and *tail*, moreover they are also offered as aggregated in *both* prediction. Further division is based on the type of tie-breaking policy used.

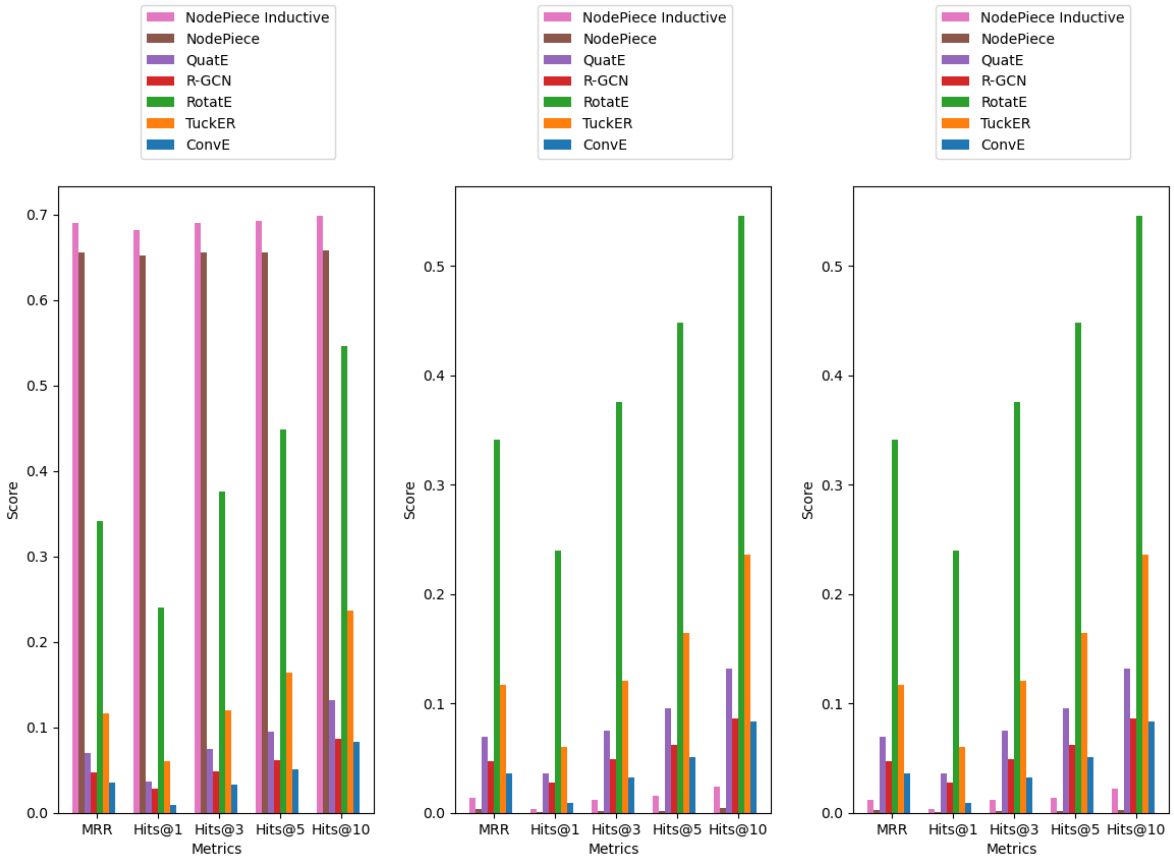
Head Prediction

The results of head prediction under the traditional prediction approach can be seen in Figure 6.1. In this case the metrics include MRR and H@K with $k = 1, 3, 5, 10$.

All models achieve the same performance in the different scenarios, except for NodePiece [14]. In fact, both transductive and inductive versions achieve better results when *min* policy is used. All models significantly improve the H@K metric when k increases. However, the performance of NodePiece [14] despite improving, achieves only slightly better performance.

According to all metrics, the best models are NodePiece [14] in the inductive version, followed by its transductive version and from RotatE [52] with *min* policy. Instead, with

avg, and *max* policies, the best model is RotatE [52], followed by TuckER [4] and QuatE [67].



(a) Models' head prediction performance on min policy.

(b) Models' head prediction performance on avg policy.

(c) Models' head prediction performance on max policy.

Figure 6.1: Models' head prediction performance on static modelling.

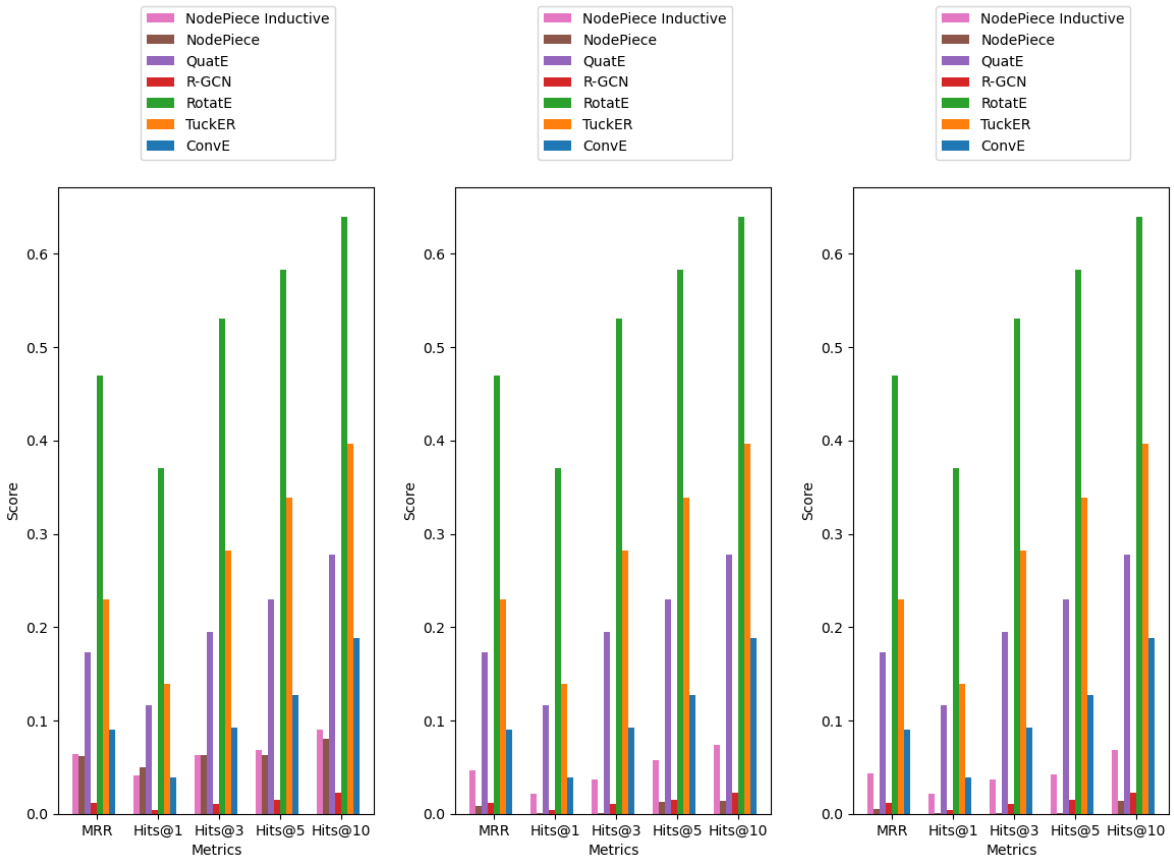
Tail Prediction

The results of tail prediction under the traditional prediction approach can be seen in Figure 6.2. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, as it is for head prediction.

According to all metrics and policies, the best model is RotatE [52], followed by TuckER [4] and QuatE [67].

All models significantly improve the H@K metric when k increases. Usually, their relative score remains the same, but with the *min* policy, NodePiece [14] surpasses the transductive version when k increases.

Moreover, the performance of NodePiece [14] is the only one that changes according to the policy. In particular, the performance of the inductive version performs slightly worse on *avg*, and *max* policy if compared with *min* policy, while the transductive version, follows the same pattern, but the drop in performance is much larger.



(a) Models' tail prediction performance on min policy.

(b) Models' tail prediction performance on avg policy.

(c) Models' tail prediction performance on max policy.

Figure 6.2: Models' tail prediction performance on static modelling.

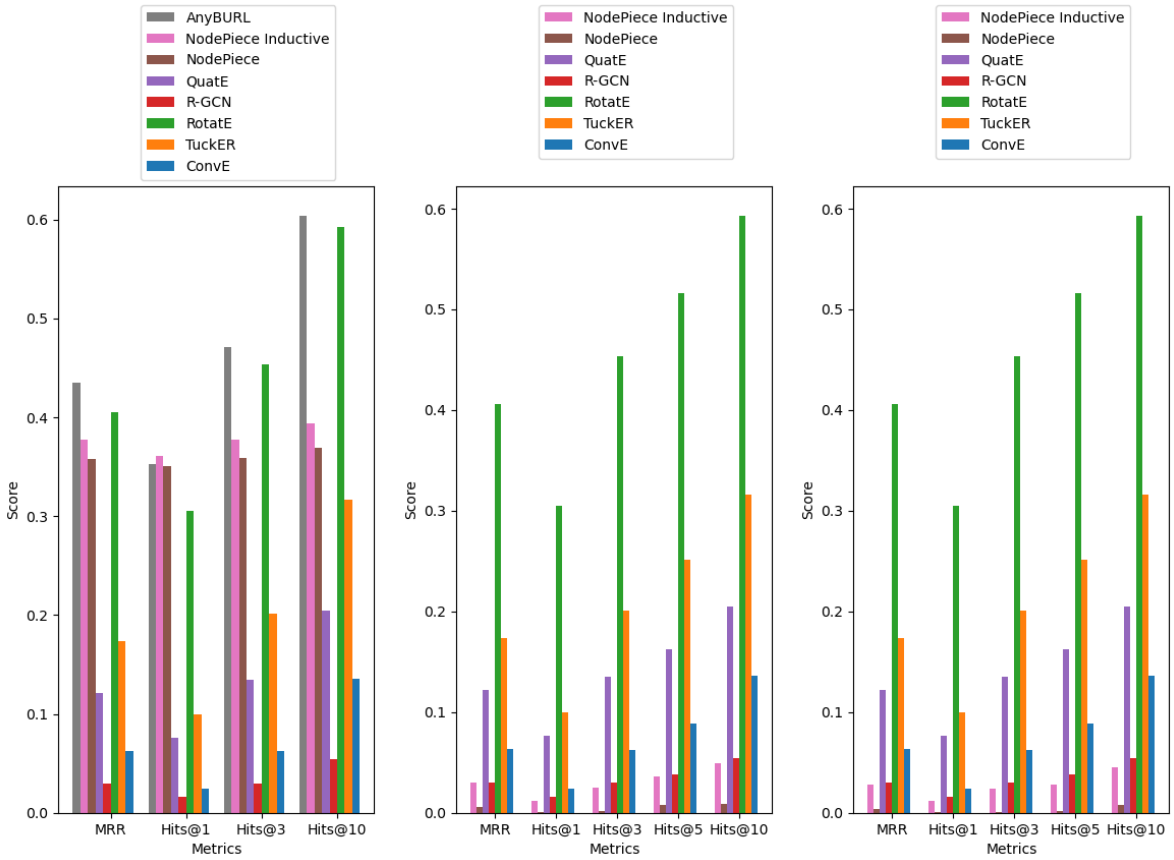
Both Prediction

The results of both predictions under the traditional prediction approach can be seen in Figure 6.3. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, for *avg* and *max* policy, while *H@5* is not present when using *min* policy, since the custom software implementation of AnyBURL [35] did not support it. Moreover, AnyBURL performance, which where not showed before, are present only when using *min* policy for the same reason.

As expected, the results of both predictions are the average of head and tail predictions. Therefore all the previous considerations on the increase of performance with k and the behaviour of the models are the same. In particular, NodePiece achieves good performance with *min* policy and gets worse when using other policies, with its transductive version that has a bigger downfall.

For *avg*, and *max* policies, and all metrics the best model is RotatE [52], followed by TuckER [4], and QuatE [67]. Moreover, the relative positions remain the same across all metrics.

The situation on *min* policy is a bit different. AnyBURL [35] achieve the best results on all metrics, except for Hits@1, where it is outperformed by NodePiece [14] Inductive, and it is slightly better than the transductive version. However, AnyBURL [35] is the best in this setting among the transductive models. According to all other metrics, the second best model is RotatE [52].



(a) Models' both prediction performance on min policy.

(b) Models' both prediction performance on avg policy.

(c) Models' both prediction performance on max policy.

Figure 6.3: Models' both prediction performance on static modelling.

6.1.2. Approach 1.2: Trace-based prediction

The performance of the trace-based prediction approach can be seen in Figure 6.4. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, for *avg* and *max* policy, while *H@5* is not present when using *min* policy, since the custom software implementation of AnyBURL [35] did not support it. Moreover, the results are not split according to *head*, and *tail* prediction as did before, but they are aggregated due to Pykeen limitations.

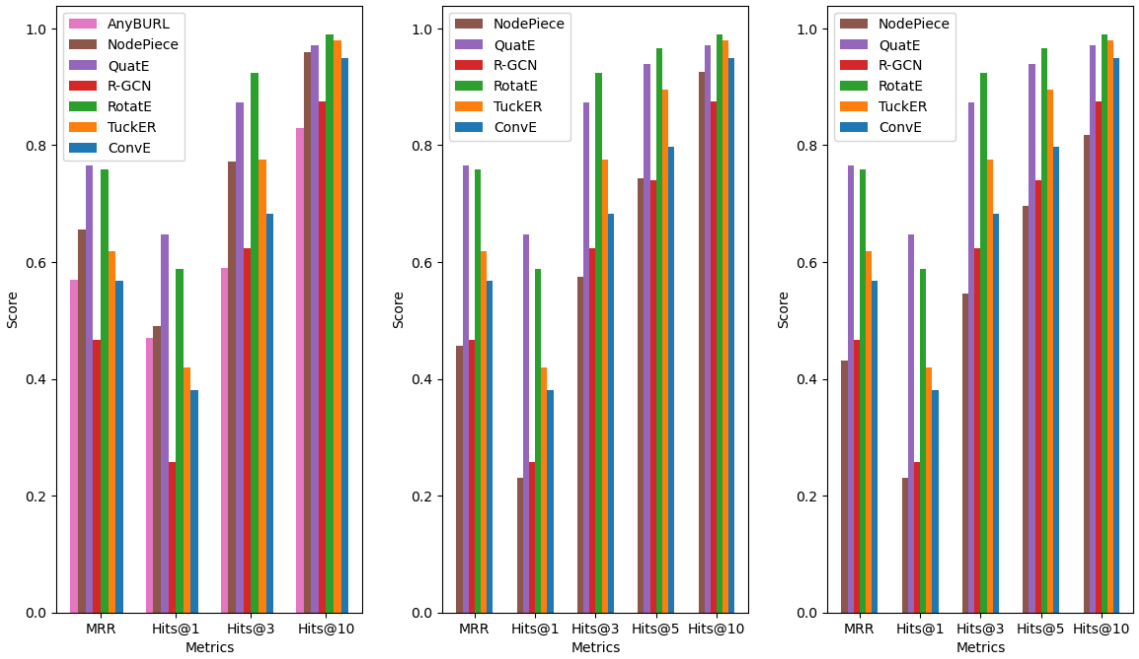
In this case, when using the *min* policy, the relative positions of the models change based on the metric used. According to MRR and Hits@1 the best model is QuatE [67], followed by RotatE [52], and NodePiece [14], and the worst performing model is R-GCN [48].
According to Hits@3, the best-performing model is RotatE [52], followed by QuatE [67], and the worst-performing model is AnyBURL [35].
On Hits@10, the situation is very similar, but the second place is taken by TuckER [4], however, all the performances of the top 5 models are comparable.

When using the *avg*, and *max* policies the results are the same, with the exception of NodePiece [14] which is always the worst-performing model.

Lastly, there are the results of topology-based methods described in Chapter 2: Common Neighbours [32], Sorensen Index [51], Salton Index [46], Jaccard Index [21], Resource Allocation [69], Adamic Adar [1], Katz Index [25], and LHN Index [30]. As expected, their performance is bad. They all achieve the same result.

Table 6.1: Topology-based methods' use-case performance on min policy and static modelling.

Models	**MRR**	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
All investigated models	0.019873	0.0	0.0	0.029211	0.047712



(a) Models' use-case performance on min policy. (b) Models' use-case performance on avg policy. (c) Models' use-case performance on max policy.

Figure 6.4: Models' use-case performance on static modelling.

6.2. Case 2: IO Modelling

In this section, the results of IO modelling are reported. They include the *traditional*, and *trace-based* prediction approach. The results obtained are similar, but slightly different from the *static* modelling case. Therefore, general considerations are the same and will not be repeated, but relative positions may change.

6.2.1. Approach 2.1: Traditional prediction

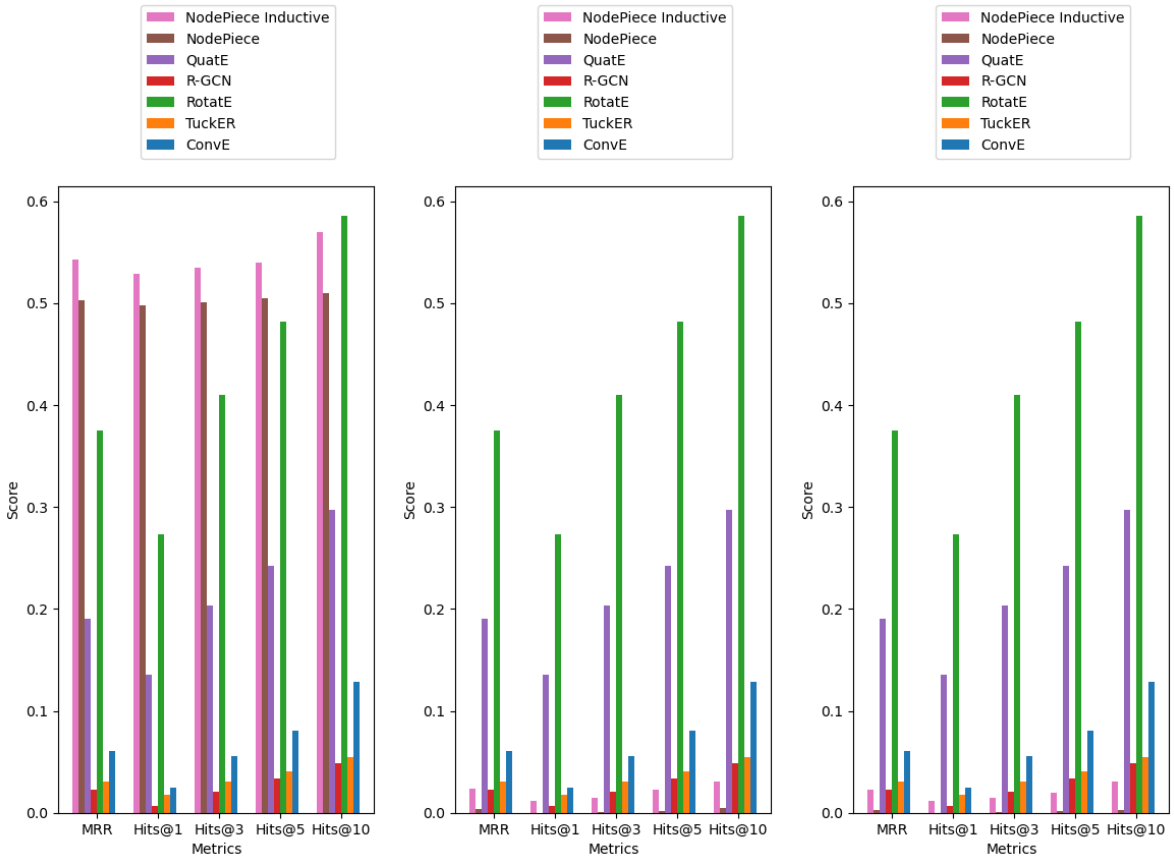
Here are presented the results divided on the type of prediction: *head*, and *tail*, moreover they are also offered as aggregated in *both* prediction. Further division is based on the type of tie-breaking policy used.

Head prediction

The results of head prediction under the traditional prediction approach can be seen in Figure 6.5. In this case the metrics include MRR and H@K with $k = 1, 3, 5, 10$.

RotatE [52] is the best model on *avg*, and *max* policies on all metrics, followed by QuatE

[67], and ConvE [9]. Instead, on *min* policy, the best models are NodePiece [14] in the inductive version, followed by its transductive version and from RotatE [52], however according to *H@10*, the RotatE [52] is able to achieve the best performance.



(a) Models' head prediction performance on min policy. (b) Models' head prediction performance on avg policy. (c) Models' head prediction performance on max policy.

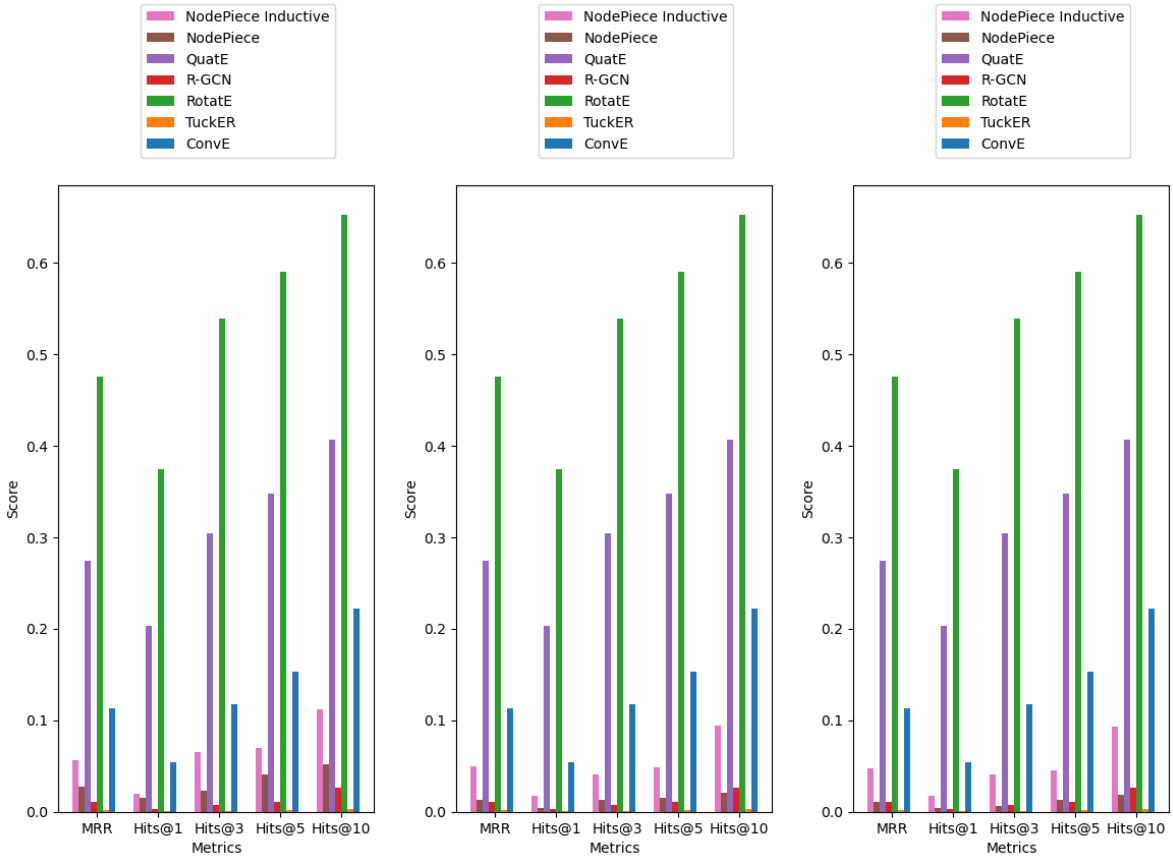
Figure 6.5: Models' head prediction performance on IO modelling.

Tail prediction

The results of tail prediction under the traditional prediction approach can be seen in Figure 6.6. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, as it is for head prediction.

According to all metrics and policies, the best model is RotatE [52], followed by QuatE [67], and ConvE [9].

Moreover, the performance of NodePiece [14] is the only one that has a peek when using *min* policy.



(a) Models' tail prediction performance on min policy.

(b) Models' tail prediction performance on avg policy.

(c) Models' tail prediction performance on max policy.

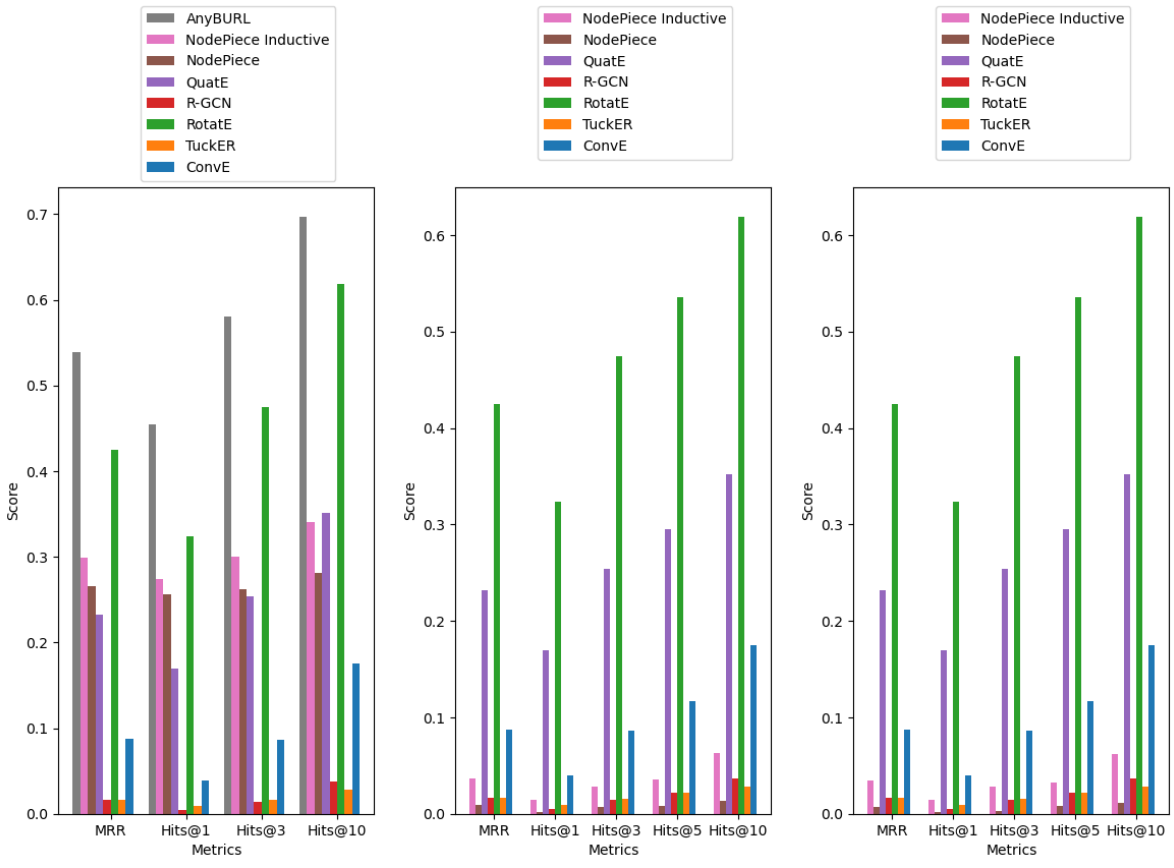
Figure 6.6: Models' tail prediction performance on IO modelling.

Both prediction

The results of both predictions under the traditional prediction approach can be seen in Figure 6.7. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, for *avg* and *max* policy, while *H@5* is not present when using *min* policy, since the custom software implementation of AnyBURL [35] did not support it. Moreover, AnyBURL performance, which where not showed before, is present only when using *min* policy for the same reason.

For *avg*, and *max* policies, and all metrics the best model is RotatE [52], followed by QuatE [67], and ConvE [9].

On *min* policy, AnyBURL [35] achieves the best results on all metrics. According to all metrics, the second-best model is RotatE [52]. The third-best model is NodePiece [14], in all metrics but *H@10*, where it is outperformed by QuatE [67].



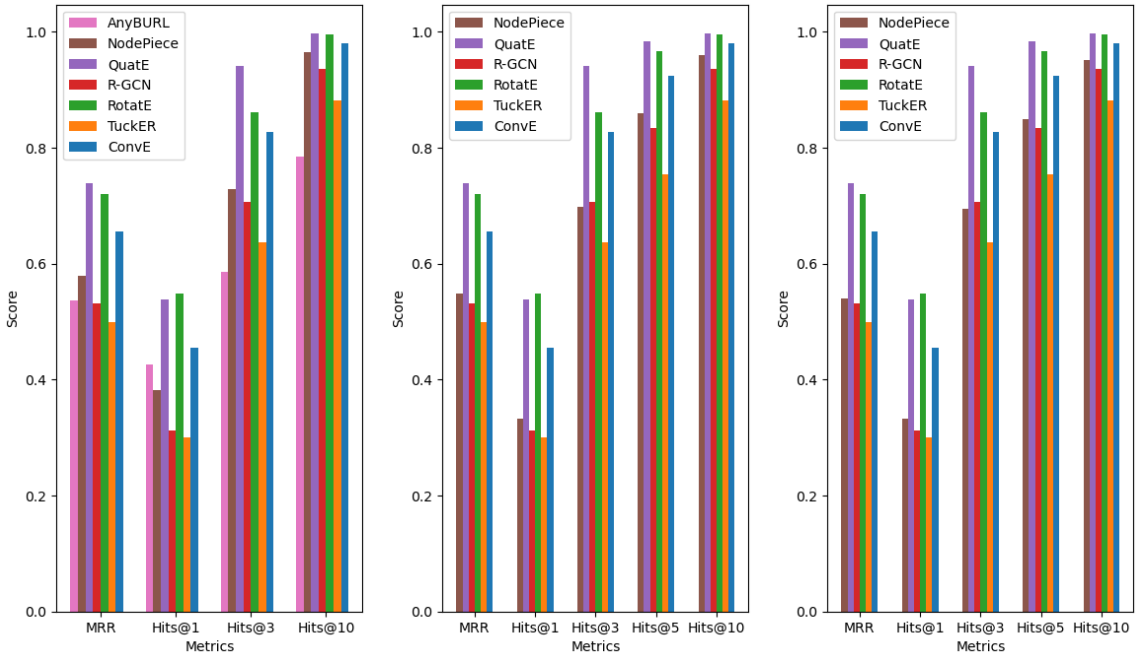
(a) Models' both prediction performance on min policy. (b) Models' both prediction performance on avg policy. (c) Models' both prediction performance on max policy.

Figure 6.7: Models' both prediction performance on IO modelling.

6.2.2. Approach 2.2: Trace-based prediction

The performance of the trace-based prediction approach can be seen in Figure 6.8. The results of Topology based methods have not been computed since they do not take into account the relation type, and therefore their behaviour is not affected by the modelling type. In this case, the metrics include MRR and H@K with $k = 1, 3, 5, 10$, for *avg* and *max* policy, while *H@5* is not present when using *min* policy, since the custom software implementation of AnyBURL [35] did not support it. Moreover, the results are not split according to *head*, and *tail* prediction as did before, but they are aggregated due to Pykeen limitations.

The relative position of the models remains the same regardless of the policy used. According to all metrics, except for *H@1*, the best model is QuatE [67], followed by RotatE [52], and [9]. Considering *H@1*, the position of QuatE[67], and RotatE [52] is swapped.



(a) Models' use-case performance on min policy.

(b) Models' use-case performance on avg policy.

(c) Models' use-case performance on max policy.

Figure 6.8: Models' use-case performance on IO modelling.

6.3. Additional Requirements

In this section, the requirements of inference time, memory usage, and training time have been reported.

6.3.1. Case 1: Static Modelling

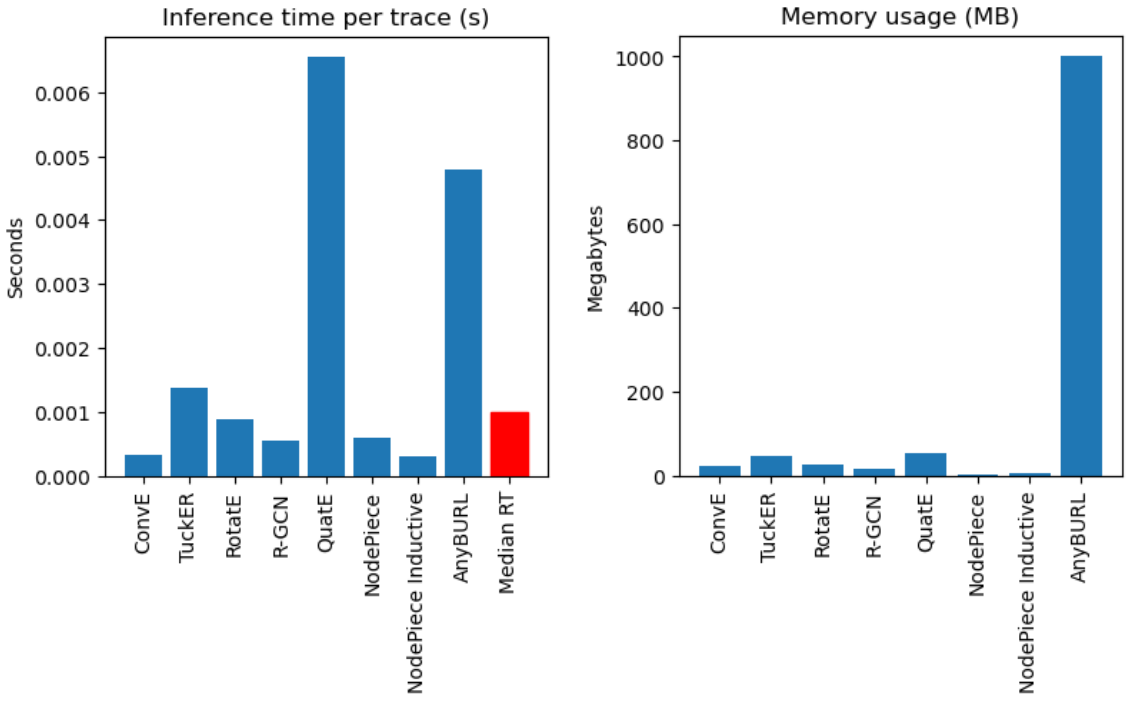
The performance of the models on the additional requirements of inference time per trace and memory usage are shown in Figure 6.9. The number of parameters can be seen in Figure 6.10b. The training time when using the entire dataset and when using the 29% of the train data can be seen in Figure 6.10a.

The best model for inference time is the inductive version of NodePiece [14], followed by ConvE [9]. AnyBURL [35], QuatE [67], and TuckER [4] take more time for prediction than the median response time of the request threshold. However, it is important to state that AnyBURL [35] runs on Java, and therefore it is not really comparable with the other models.

The transductive version of NodePiece [14] is the model that uses less memory, followed by its inductive version, which uses 10 times more memory, which in turn, is followed by all the other models that use 10 times more memory. An exception is AnyBURL [35], which uses a lot more memory since 1 GB of RAM is the smallest amount of ram that can be used according to the proprietary program.

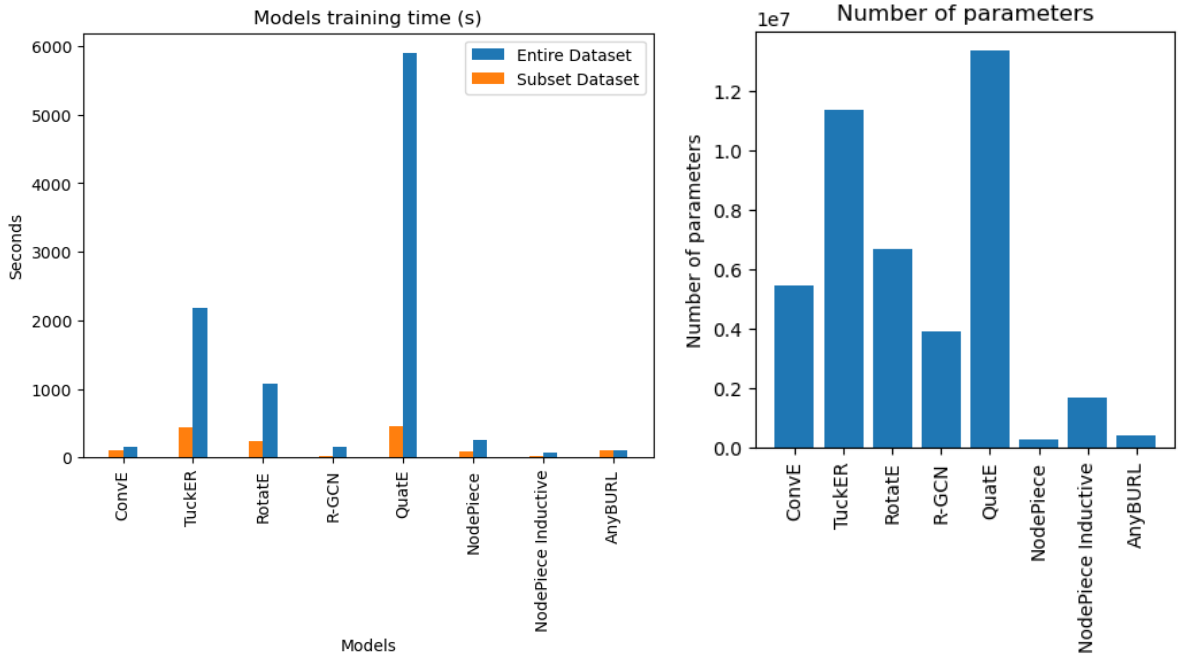
Except for AnyBURL [35], the number of parameters of the model is proportional to the memory usage with NodePiece [14] being the ones that have fewer parameters and QuatE [67] being the one that has most.

The best model per training duration is NodePiece inductive [14], followed by AnyBURL [35], and ConvE [9]. The worst-performing model according to this metric si QuatE [67], which took more than double the amount of time to train the second worst-performing model.
When using just a subset of the entire data, the relative position of some close models changed, in particular, R-GCN [48] and NodePiece [14] significantly decrease their training time, and outperform ConvE [9]. However, when considering the extremes, the results are the same. Nevertheless, slower models have an increase in the training time which is more than linear.
AnyBURL [35] training time did not change since it is a hyperparameter that can be specified.



(a) Models' inference time per triplet. (b) Models' memory usage in bytes.

Figure 6.9: Additional requirements performance on static modelling.

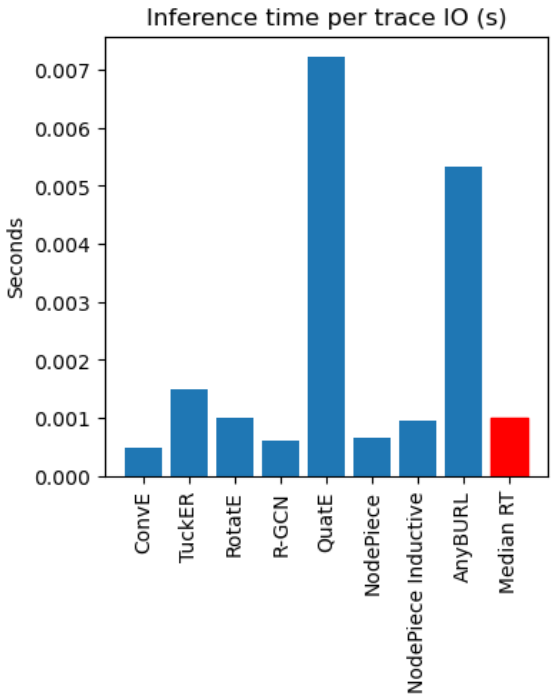


(a) Models' training time. (b) Models' number of parameters.

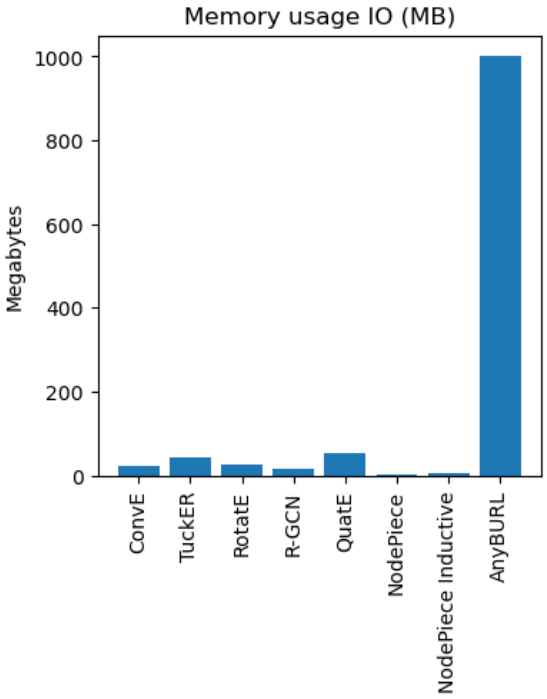
Figure 6.10: Additional requirements on static modelling.

6.3.2. Case 2: IO Modelling

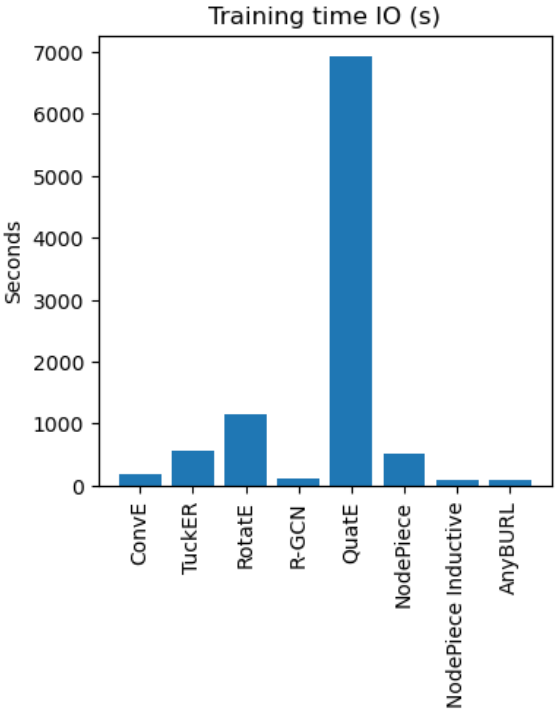
The performance of the models on the additional requirements of are shown in Figure 6.11. All the considerations done for the *static* modelling case, still hold. The difference is not significant, but the results are reported for correctness.



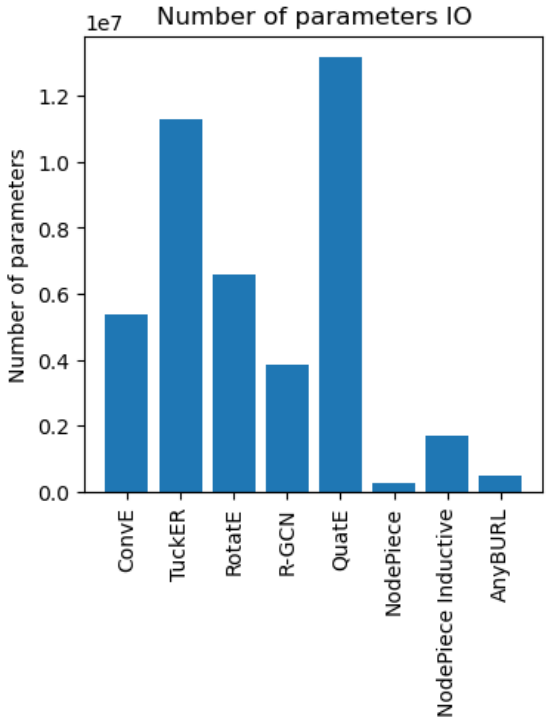
(a) Models' inference time per triplet.



(b) Models' memory usage in bytes.



(c) Models' training time.



(d) Models' number of parameters.

Figure 6.11: Additional requirements performance on IO modelling.

6.4. Dynamic Sampling

The last step of this work is a simulation of an application of the results of this thesis. It is an additional step that does not directly answers the research question.

The example chosen for this research is the integration of a link prediction method with a dynamic sampling microservice tool. The idea is that a tracing tool can assign a different sample rate to different microservices by using some criteria such as priority or time complexity. For example, when considering the latter scenario, traces related to light web services would be collected with higher probability, while more complex microservices would be reconstructed using the investigated methods. In this way, it would be possible to reduce the tracing overhead.

The model chosen for the simulation is RotatE [52], as will be motivated in Chapter 7.

The tracing tool assigns sample rates as follows:

- The sample rate is assigned independently to each microservice m, and it is the probability of sampling or not an outgoing RPC from the microservice m.
- When a microservice has not been sampled yet, its associated probability is 1. In this way, the transductive property is ensured.
- When a microservice has been sampled for the first time, its associated probability changes, and it is taken from a uniform distribution $U(0,01;0,1)$

The model is trained following the *static* modelling, and it is tested with the *trace-based* prediction since some calls would be missed otherwise. Due to the sampling, the final dataset had just 41384 triplets, which is 25.5% smaller than the final dataset under static modelling without sampling. Therefore the training has been affected. However, the dataset used for evaluating the *trace-based* prediction is the same as the not sampled comparison.

6.4.1. Results

The accuracy performance of *trace-based* prediction on *avg* policy of RotatE [52] simulating a dynamic sampling scenario is shown in Table 6.2. The additional requirements performance are shown in Table 6.3. They are very similar to the ones obtained in the main comparison.

Table 6.2: Dynamic Sampling trace-based prediction performance on average policy

MRR	Hits@1	Hits@3	Hits@5	Hits@10
0.70	0.52	0.87	0.96	0.99

Table 6.3: Dynamic Sampling additional requirements

Training time	Inference time per trace	Memory usage	Number of parameters
942.81 s	0.00098 s	26.66 MB	6665200

7 | Conclusions

This chapter describes in more detail and discusses the findings of the master thesis project. Moreover, it includes suggestions for future work. Lastly, there are also some final words.

7.1. Discussion

Here there are the considerations of the results presented in the previous chapter. They are divided based on the modelling type used and the prediction scope approach used. The additional requirements considerations are valid for both approaches since the results are almost identical.

7.1.1. Case 1: Static Modelling

This section presents the considerations on the static modelling case.

Approach 1.1: Traditional prediction

While RotatE [52], QuatE [67], TuckER [4], ConvE [9], and R-GCN have coherent performance in the different settings and under different policies, and it is easy to state that RotatE is the best model, it is not immediate to draw a conclusion when comparing with NodePiece [14], and AnyBURL [35], due to their different behaviour.

In particular, NodePiece [14] is heavily influenced by the prediction policy used. Its optimal results on *min* policy may lead to unnecessary praise. The *min* or *optimal* policy assigns the lower rank when there are ties. Therefore a trivial model that gives the same score to all available triplets will have a perfect score. According to this consideration and the below par results achieved by NodePiece [14] in the other policies, it is clear that the model does not have good accuracy and it is not able to rank properly the results. However, while it may seem that the results obtained are in contradiction with previous studies, it may not be the case. In particular, the results obtained by Galkin et al. [14] are similar to the ones obtained when using the *min* policy. Moreover, it is possible that

using different hyperparameters would have led to better results.

For the same reason, and due to the fact that the results are available only when using the *optimal* policy in *both* prediction, it is not possible to understand properly how is the performance of AnyBURL [35] when compared with other methods.

The outcome of this analysis is in contradiction with previous studies. In particular, according to Rossi et al. [29], and Balazevic et al. [4], TuckER [4] outperforms RotatE [52] in most of the cases; according to Zhang et al. QuatE [67] outperforms RotatE [52]. However, the difference in the performance according to the previously mentioned papers has been minimal, and therefore the results obtained by this research could be explained by the usage of a different dataset. In particular, the dataset used in this thesis is smaller from all perspectives (number of entities, number of relation types, and number of triplets) by a factor of 10 than the ones used in the literature such as FB15k [7], WN18 [15], and YAGO3-10 [33].

Approach 1.2: Trace-based prediction

In this case, the metrics have generally a higher value, since the number of output entities is constrained to the ones that are actually in the trace. In particular, *Hits@10* is very high since there is a finite number of combinations among the available entities.

For this reason, *MRR*, and *Hits@K* with low k values are more meaningful than the other metrics. Therefore the best model in the use-case scenario is QuatE [67]. It may be due to the fact that its rich embedding with a high number of parameters allows for better discrimination among the single traces.

The previous consideration still holds when considering the *avg* policy, which is, arguably, the more relevant policy for a conclusive comparison, as explained by the previous consideration for NodePiece [14], and supported by Rossi et al. [29].

Topology-based methods achieve very poor results even with *min* policy since they need that two nodes are already connected through a path, in order to predict an edge between them. Therefore it is not possible to predict a link to a microservice that has not been called yet. Moreover, the reason they achieve the same results is probably due to the fact that there are very few traces that have just one connected component, and in those cases, they are able to identify properly the missing link since it may be the only option and the score assigned to a possible triplet is always proportional of the common neighbours of the two nodes involved.

The results obtained in this case cannot be compared with previous studies since the

output has been constrained, in order to have a meaningful result for the use case.

7.1.2. Case 2: IO Modelling

The results obtained are very similar to the *static* modelling case, therefore most of the considerations, in particular the general ones still hold.

Approach 2.1: Traditional prediction

The main difference with the *static* modelling case is that TuckER [4] is not performing well, in fact, it is one of the worst-performing models. However, it is not clear why this happens. Moreover, QuatE [67] seems to have improved performance, and it is coherent with the previous considerations. However, it makes sense that a more complex model is able to work better when there is more complexity in the data.

Approach 2.2: Trace-based prediction

Following the same reasoning as before, and therefore giving more importance to *MRR*, and *H@1*, it is not clear which is the best model between RotatE [52], and QuatE [67]. However, despite having slightly different results from the static modelling scenario, they both make sense, since the difference in accuracy between the two models is very low. Therefore it is possible that based on the subset of data used, or based on the way the problem is modelled, one model outperforms the other by a low delta.

7.1.3. Additional Requirements

When considering memory usage, it is not possible to compare AnyBURL [35] with other models since it is not clear how much memory the model actually needs. Since its memory usage value is an outlier, it is likely that it does not need the entire amount of space reserved. However, it is reasonable to think that the amount of memory required is higher than the other models since it is storing rules and not a shallow embedded representation.

The other models, instead, have a smaller memory usage, that has a more similar order of magnitude. However, there are still some differences despite having the same embedding size. It is possible to understand the reason by looking at the number of parameters per model. In particular, QuatE [67], and TuckER [4] achieve the worst results among shallow embedding models, and it is due to the fact that they learn a very rich representation and therefore have a large number of parameters. On the other hand, NNs based models are the ones that use less memory and have a lower number of parameters. In particular,

NodePiece is the best model and it makes sense since it is an inductive model, and it does not need to store a representation for each entity, but it relies on the anchors.

Considering the inference time per trace, the three models that are above the threshold are TuckER [4], QuatE [67], and AnyBURL [35]. While it is reasonable to expect a long inference time for the latter model, it is surprising that a rule-based method is still faster than an embedding-based method. However, they have been monitored on two different platforms, therefore it is not possible to compare them. While it is true that the slower models are the ones with a more rich entity representation, it is not always true that having a simpler representation makes faster predictions. In particular R-GCN [48], and NodePiece [14] transductive are slower than ConvE despite having a lower memory usage. This could be caused by the structure of the models, it is not surprising that a naive model such as R-GCN [48] is not very fast.

The results of the training time seem to be strictly related to the ones of inference time, therefore the same considerations apply. However, the relative difference between training times in the two sets seems to follow a relation which is more than linear.

Compared with previous studies such as Rossi et al. [29], the results of the training are a bit different since they did not use early stopping. In particular, according to Rossi et al. [29] TuckER [4] has been generally faster to train than ConvE [9], and RotatE [52]. However, the results obtained by this thesis show that the use of early stopping seems a better way to assess the training time, and does not have a negative impact on performance. In fact, RotatE [52] trained faster than TuckER [4] and had better accuracy.

Moreover, the results show that despite Zhang et al. [67] claim that QuatE [67] inference time is linear in the size of the embedding space as it is for RotatE [52], the actual time required for prediction may be so different that the model cannot be used in scenarios where the inference time is critical.

7.2. Final Comparison

This section aims to answer the research question by summarizing the findings for each model investigated:

- RotatE [52] can be considered the best model since it has optimal performance from an accuracy perspective and good enough performance when considering the additional requirements. In particular, its inference time per trace is lower than the threshold. It is well suited for an application.

- QuatE [67] has often achieved good accuracy performance, but it achieved the worst performance in training time, and inference time per trace, moreover it is the second worst performing model when considering the memory usage. Therefore it does not seem particularly well suited for an application.
- TuckER [4] has an accuracy performance that is comparable with QuatE [67] and it achieves slightly better performance on the additional requirements. Therefore the conclusion is the same as QuatE [67].
- ConvE [9] was at most the third best model when considering the accuracy, however, there was a large gap from the best model. The additional requirements performance have been really good. Therefore, if the model had better accuracy performance, which could potentially be achieved by hyperparameter tuning it would be perfect for an application.
- R-GCN [48] accuracy performance where below par, and its additional requirements performance where average. Therefore it does not seem particularly well suited for an application.
- NodePiece [14] has often been the best model when considering the additional requirements, but it has achieved below-par performance when considering the accuracy. Therefore it does not seem particularly well suited for this use case, but since it is very fast, it could be used for a problem where its accuracy performance is better.
- Anyburl [35] has achieved peculiar performance from the additional requirement perspective, in particular, it had a low training time, but a high inference time and memory usage. Moreover, as previously explained, it is not possible to really compare its accuracy performance with the other models. Therefore it is not really possible to draw a conclusion and and this model requires further investigation.

7.3. Dynamic Sampling

RotatE [52] seemed to be the optimal choice for the additional simulation considering the accuracy performance in both modelling approaches, an inference time per trace lower than the threshold, low memory usage and good scores with the additional requirements.

The results obtained are very similar to the ones of the main analysis despite using 25.5% less data. They show that an approach like this can be used for a real-case scenario. However, it is important to adjust the sampling probability in order to collect all entities

to ensure the transductive property. This is the bigger limitation since it may require sampling for a long time before obtaining all the required data.

7.4. Future Work

This study raises several intriguing questions that could be addressed in future research.

In particular, the two prediction scope approaches have been necessary since models are trained and tested on a big single graph, while it would be interesting to develop new models that can learn on multiple small graphs.

Another direction could be studying according to the same metrics models that take into account time dependency, such as HyTE [31], and TAGCN [11].

Moreover, it could be interesting to develop a Python-based version of AnyBURL [35], and assess its performance according to the settings and policies that could not be addressed.

Furthermore, performing hyperparameter tuning is definitely a good idea for future work, since it may lead to better results.

Lastly, the results of this research can lay the foundation for the integration of link prediction methods in microservice tracing applications. An idea could be to model the data following the *IO* approach or in a different way and see which are the results.

7.5. Final Words

This thesis compared RotatE [52], QuatE [67], TuckER [4], ConvE [9], R-GCN [48], NodePiece [14], and AnyBURL [35] on link prediction, using as accuracy metrics MRR, and H@K. Moreover, also the training time, the inference time and the memory usage were compared. From the available data, two different knowledge graphs have been built: the static model, and the IO model. The former did not take into account the temporal information, while the latter did. Moreover, the models have been evaluated on two different prediction scopes: the classic approach that followed standard results in the literature, and the use-case approach that mimicked a real-case scenario.

The main takeout of this analysis is that rich embeddings models such as QuatE [67], and TuckER [4], have a long inference time, and training time, moreover, they have a high memory usage. Therefore, they cannot be used in a real-case scenario, where time is important. Instead, a model like RotatE [52], is able to achieve good performance from an accuracy perspective and it is also able to achieve average results in the previously

defined requirements. Therefore it is an appropriate choice to build an application on.

Furthermore, methods that do not learn an embedding for each entity, such NodePiece [14] while having generally a fast inference time, seem to not achieve good accuracy performance when compared to other methods. In particular, when considering NodePiece [14], it is true that its inductive version performed generally better than the transductive one, however, its performance was worse than the other transductive model. Therefore, in the microservice scenario, it seems better to use a transductive approach.

Moreover, the way the problem is modelled may have a big impact on the accuracy performance of the models, as it happens for TuckER [4]. Therefore, building the knowledge graph in the proper way has a significant effect on the performance of the application.

Lastly, the dynamic sampling simulation shows that the results obtained are similar to the *trace-based* prediction case, and therefore the work is going to the right direction and can be used to reduce the overhead caused by tracing tools.

Bibliography

[1] Adamic, Lada A and Adar, Eytan. "Friends and neighbors on the Web". In: *Social Networks* 25.3 (July 2003), pp. 211–230. ISSN: 03788733. DOI: 10.1016/S0378-8733(03)00009-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378873303000091 (visited on 02/04/2023).

[2] *Alibaba Cluster Trace Program*. original-date: 2017-09-05T03:16:34Z. Mar. 14, 2023. URL: https://github.com/alibaba/clusterdata (visited on 03/14/2023).

[3] *B-MEG | Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*. URL: https://dl-acm-org.focus.lib.kth.se/doi/10.1145/3491204.3527494 (visited on 01/21/2023).

[4] Balažević, Ivana, Allen, Carl, and Hospedales, Timothy M. "TuckER: Tensor Factorization for Knowledge Graph Completion". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 5184–5193. DOI: 10.18653/v1/D19-1522. arXiv: 1901.09590[cs,stat]. URL: http://arxiv.org/abs/1901.09590 (visited on 12/05/2022).

[5] Besta, Maciej, Peter, Emanuel, Gerstenberger, Robert, Fischer, Marc, Podstawski, Michał, Barthels, Claude, Alonso, Gustavo, and Hoefler, Torsten. *Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries*. Nov. 4, 2022. arXiv: 1910.09017[cs]. URL: http://arxiv.org/abs/1910.09017 (visited on 02/23/2023).

[6] Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, and Taylor, Jamie. "Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 1247–1250. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376746. URL: https://dl.acm.org/doi/10.1145/1376616.1376746 (visited on 05/18/2023).

[7] Bordes, Antoine, Usunier, Nicolas, Garcia-Duran, Alberto, Weston, Jason, and Yakhnenko, Oksana. "Translating Embeddings for Modeling Multi-relational Data". In: (), p. 10.

[8] Cai, Jin-Yi, Fürer, Martin, and Immerman, Neil. "An optimal lower bound on the number of variables for graph identification". In: *Combinatorica* 12.4 (Dec. 1, 1992), pp. 389–410. ISSN: 1439-6912. DOI: 10.1007/BF01305232. URL: https://doi.org/10.1007/BF01305232 (visited on 12/05/2022).

[9] Dettmers, Tim, Minervini, Pasquale, Stenetorp, Pontus, and Riedel, Sebastian. *Convolutional 2D Knowledge Graph Embeddings*. July 4, 2018. arXiv: 1707.01476[cs]. URL: http://arxiv.org/abs/1707.01476 (visited on 12/06/2022).

[10] Dong, Xin, Gabrilovich, Evgeniy, Heitz, Geremy, Horn, Wilko, Lao, Ni, Murphy, Kevin, Strohmann, Thomas, Sun, Shaohua, and Zhang, Wei. "Knowledge vault: a web-scale approach to probabilistic knowledge fusion". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '14: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York New York USA: ACM, Aug. 24, 2014, pp. 601–610. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623623. URL: https://dl.acm.org/doi/10.1145/2623330.2623623 (visited on 05/18/2023).

[11] *Enhance Temporal Knowledge Graph Completion via Time-Aware Attention Graph Convolutional Network | SpringerLink*. URL: https://link-springer-com.focus.lib.kth.se/chapter/10.1007/978-3-031-26390-3_8 (visited on 04/12/2023).

[12] Frasconi, P., Gori, M., and Sperduti, A. "A general framework for adaptive processing of data structures". In: *IEEE Transactions on Neural Networks* 9.5 (Sept. 1998). Conference Name: IEEE Transactions on Neural Networks, pp. 768–786. ISSN: 1941-0093. DOI: 10.1109/72.712151.

[13] Galkin, Mikhail, Berrendorf, Max, and Hoyt, Charles Tapley. *An Open Challenge for Inductive Link Prediction on Knowledge Graphs*. Apr. 18, 2022. arXiv: 2203.01520[cs]. URL: http://arxiv.org/abs/2203.01520 (visited on 02/27/2023).

[14] Galkin, Mikhail, Denis, Etienne, Wu, Jiapeng, and Hamilton, William L. *NodePiece: Compositional and Parameter-Efficient Representations of Large Knowledge Graphs*. Feb. 1, 2022. DOI: 10.48550/arXiv.2106.12144. arXiv: 2106.12144[cs]. URL: http://arxiv.org/abs/2106.12144 (visited on 12/06/2022).

[15] Glorot, Xavier, Bordes, Antoine, Weston, Jason, and Bengio, Yoshua. *A Semantic Matching Energy Function for Learning with Multi-relational Data*. Mar. 21, 2013. DOI: 10.48550/arXiv.1301.3485. arXiv: 1301.3485[cs]. URL: http://arxiv.org/abs/1301.3485 (visited on 04/16/2023).

[16] Håkansson, Anne. "Portal of Research Methods and Methodologies for Research Projects and Degree Projects". In: *Computer Engineering* (2013).

[17] Hamilton, William L. *Graph representation learning*. Synthesis lectures on artificial intelligence and machine learning ; Number 46. Publication Title: Graph represen-

tation learning. Place of publication not identified: Morgan & Claypool Publishers, 2020. ISBN: 1-68173-964-X.

[18] Hamilton, William L., Ying, Rex, and Leskovec, Jure. *Inductive Representation Learning on Large Graphs*. Sept. 10, 2018. DOI: 10.48550/arXiv.1706.02216. arXiv: 1706.02216[cs,stat]. URL: http://arxiv.org/abs/1706.02216 (visited on 12/05/2022).

[19] Hamilton, William L., Ying, Rex, and Leskovec, Jure. *Representation Learning on Graphs: Methods and Applications*. Apr. 10, 2018. DOI: 10.48550/arXiv.1709.05584. arXiv: 1709.05584[cs]. URL: http://arxiv.org/abs/1709.05584 (visited on 01/24/2023).

[20] *How Google's Knowledge Graph works - Knowledge Panel Help*. URL: https://support.google.com/knowledgepanel/answer/9787176?hl=en (visited on 02/08/2023).

[21] Jaccard, Paul. "Étude comparative de la distribution florale dans une portion des Alpes et du Jura". In: (Jan. 1, 1901). Publisher: Imprimerie Corbaz & Comp. DOI: 10.5169/seals-266450. URL: https://www.scienceopen.com/document?vid=01a9bb0e-a14c-461f-85c9-f407259515dc (visited on 02/04/2023).

[22] Jackson, Matthew O. "Social and Economic Networks". In: ().

[23] *Jaeger: open source, end-to-end distributed tracing*. URL: https://www.jaegertracing.io/ (visited on 01/21/2023).

[24] Ji, Guoliang, He, Shizhu, Xu, Liheng, Liu, Kang, and Zhao, Jun. "Knowledge Graph Embedding via Dynamic Mapping Matrix". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. ACL-IJCNLP 2015. Beijing, China: Association for Computational Linguistics, July 2015, pp. 687–696. DOI: 10.3115/v1/P15-1067. URL: https://aclanthology.org/P15-1067 (visited on 01/23/2023).

[25] Katz, Leo. "A new status index derived from sociometric analysis". In: *Psychometrika* 18.1 (Mar. 1, 1953), pp. 39–43. ISSN: 1860-0980. DOI: 10.1007/BF02289026. URL: https://doi.org/10.1007/BF02289026 (visited on 02/04/2023).

[26] Kazemi, Seyed Mehran and Poole, David. *SimplE Embedding for Link Prediction in Knowledge Graphs*. Oct. 25, 2018. arXiv: 1802.04868[cs,stat]. URL: http://arxiv.org/abs/1802.04868 (visited on 12/05/2022).

[27] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. arXiv: 1412.6980[cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 03/15/2023).

[28] Kipf, Thomas N. and Welling, Max. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 22, 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907[cs,stat]. URL: http://arxiv.org/abs/1609.02907 (visited on 12/05/2022).

[29] *Knowledge Graph Embedding for Link Prediction: A Comparative Analysis: ACM Transactions on Knowledge Discovery from Data: Vol 15, No 2*. URL: https://dl-acm-org.focus.lib.kth.se/doi/abs/10.1145/3424672 (visited on 12/05/2022).

[30] Leicht, E. A., Holme, Petter, and Newman, M. E. J. "Vertex similarity in networks". In: *Physical Review E* 73.2 (Feb. 17, 2006), p. 026120. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.73.026120. arXiv: physics/0510143. URL: http://arxiv.org/abs/physics/0510143 (visited on 02/11/2023).

[31] Liu, Kangzheng and Zhang, Yuhong. *A Temporal Knowledge Graph Completion Method Based on Balanced Timestamp Distribution*. Nov. 6, 2021. arXiv: 2108.13024[cs]. URL: http://arxiv.org/abs/2108.13024 (visited on 04/12/2023).

[32] Lu, Linyuan and Zhou, Tao. "Link Prediction in Complex Networks: A Survey". In: *Physica A: Statistical Mechanics and its Applications* 390.6 (Mar. 2011), pp. 1150–1170. ISSN: 03784371. DOI: 10.1016/j.physa.2010.11.027. arXiv: 1010.0725[physics]. URL: http://arxiv.org/abs/1010.0725 (visited on 12/20/2022).

[33] Mahdisoltani, Farzaneh. "A Knowledge Base from Multilingual Wikipedias – YAGO3". In: ().

[34] Martínez, Víctor, Berzal, Fernando, and Cubero, Juan-Carlos. "A Survey of Link Prediction in Complex Networks". In: *ACM Computing Surveys* 49.4 (Dec. 31, 2017), pp. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3012704. URL: https://dl.acm.org/doi/10.1145/3012704 (visited on 02/24/2023).

[35] Meilicke, Christian, Chekol, Melisachew Wudage, Ruffinelli, Daniel, and Stuckenschmidt, Heiner. "Anytime Bottom-Up Rule Learning for Knowledge Graph Completion". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Twenty-Eighth International Joint Conference on Artificial Intelligence {IJCAI-19}. Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 3137–3143. ISBN: 978-0-9992411-4-1. DOI: 10.24963/ijcai.2019/435. URL: https://www.ijcai.org/proceedings/2019/435 (visited on 04/04/2023).

[36] Newman, M. E. J. *Networks: an introduction*. OCLC: ocn456837194. Oxford ; New York: Oxford University Press, 2010. 772 pp. ISBN: 978-0-19-920665-0.

[37] Nguyen, Dai Quoc, Nguyen, Tu Dinh, Nguyen, Dat Quoc, and Phung, Dinh. "A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network". In: *Proceedings of the 2018 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). NAACL-HLT 2018. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 327–333. DOI: 10.18653/v1/N18-2053. URL: https://aclanthology.org/N18-2053 (visited on 12/06/2022).

[38] Nguyen, Hoa Xuan, Zhu, Shaoshu, and Liu, Mingming. "A Survey on Graph Neural Networks for Microservice-Based Cloud Applications". In: *Sensors* 22.23 (Jan. 2022). Number: 23 Publisher: Multidisciplinary Digital Publishing Institute, p. 9492. ISSN: 1424-8220. DOI: 10.3390/s22239492. URL: https://www.mdpi.com/1424-8220/22/23/9492 (visited on 01/17/2023).

[39] Nickel, Maximilian, Rosasco, Lorenzo, and Poggio, Tomaso. "Holographic Embeddings of Knowledge Graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Mar. 2, 2016). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v30i1.10314. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10314 (visited on 01/23/2023).

[40] *OpenTelemetry*. OpenTelemetry. URL: https://opentelemetry.io/ (visited on 02/08/2023).

[41] *OpenZipkin · A distributed tracing system*. URL: https://zipkin.io/ (visited on 02/08/2023).

[42] Padgett, John F. and Ansell, Christopher K. "Robust Action and the Rise of the Medici, 1400-1434". In: *American Journal of Sociology* 98.6 (1993). Publisher: University of Chicago Press, pp. 1259–1319. ISSN: 0002-9602. URL: https://www.jstor.org/stable/2781822 (visited on 02/03/2023).

[43] Patterson, David, Gonzalez, Joseph, Le, Quoc, Liang, Chen, Munguia, Lluis-Miquel, Rothchild, Daniel, So, David, Texier, Maud, and Dean, Jeff. "Carbon Emissions and Large Neural Network Training". In: ().

[44] Qiu, Haoran, Banerjee, Subho S, Jha, Saurabh, Kalbarczyk, Zbigniew T, and Iyer, Ravishankar K. "FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Microservices". In: ().

[45] Qiu, Juan, Du, Qingfeng, Yin, Kanglin, Zhang, Shuang-Li, and Qian, Chongshu. "A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications". In: *Applied Sciences* 10.6 (Jan. 2020). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, p. 2166. ISSN: 2076-3417. DOI: 10.3390/app10062166. URL: https://www.mdpi.com/2076-3417/10/6/2166 (visited on 01/28/2023).

[46] Salton, Gerard and McGill, Michael J. *Introduction to modern information retrieval*. McGraw-Hill computer science series. New York: McGraw-Hill, 1983. 448 pp. ISBN: 978-0-07-054484-0.

[47] Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009). Conference Name: IEEE Transactions on Neural Networks, pp. 61–80. ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605.

[48] Schlichtkrull, Michael, Kipf, Thomas N., Bloem, Peter, Berg, Rianne van den, Titov, Ivan, and Welling, Max. "Modeling Relational Data with Graph Convolutional Networks". In: *The Semantic Web*. Ed. by Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 593–607. ISBN: 978-3-319-93417-4. DOI: 10.1007/978-3-319-93417-4_38.

[49] Shuvo, G Kibria. "Tail Based Sampling Framework for Distributed Tracing Using Stream Processing". In: ().

[50] Sigelman, Benjamin H., Barroso, Luiz André, Burrows, Mike, Stephenson, Pat, Plakal, Manoj, Beaver, Donald, Jaspan, Saul, and Shanbhag, Chandan. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Google, Inc., 2010. URL: https://research.google.com/archive/papers/dapper-2010-1.pdf (visited on 02/07/2023).

[51] Sørensen, Thorvald Julius. *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. København: I kommission hos E. Munksgaard, 1948. 34 pp. URL: https://search.library.wisc.edu/catalog/9910080625002121 (visited on 02/04/2023).

[52] Sun, Zhiqing, Deng, Zhi-Hong, Nie, Jian-Yun, and Tang, Jian. *RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space*. Feb. 26, 2019. arXiv: 1902.10197[cs,stat]. URL: http://arxiv.org/abs/1902.10197 (visited on 12/05/2022).

[53] Svozil, Daniel, Kvasnicka, Vladimír, and Pospichal, Jirí. "Introduction to multilayer feed-forward neural networks". In: *Chemometrics and Intelligent Laboratory Systems* 39.1 (Nov. 1, 1997), pp. 43–62. ISSN: 0169-7439. DOI: 10.1016/S0169-7439(97)00061-0. URL: https://www.sciencedirect.com/science/article/pii/S0169743997000610 (visited on 11/15/2022).

[54] Thönes, Johannes. "Microservices". In: *IEEE Software* 32.1 (Jan. 2015). Conference Name: IEEE Software, pp. 116–116. ISSN: 1937-4194. DOI: 10.1109/MS.2015.11.

[55] Trouillon, Théo, Welbl, Johannes, Riedel, Sebastian, Gaussier, Éric, and Bouchard, Guillaume. *Complex Embeddings for Simple Link Prediction*. June 20, 2016. DOI:

10.48550/arXiv.1606.06357. arXiv: 1606.06357[cs,stat]. URL: http://arxiv.org/abs/1606.06357 (visited on 12/05/2022).

[56] Tucker, Ledyard R. "Some mathematical notes on three-mode factor analysis". In: *Psychometrika* 31.3 (Sept. 1, 1966), pp. 279–311. ISSN: 1860-0980. DOI: 10.1007/BF02289464. URL: https://doi.org/10.1007/BF02289464 (visited on 02/09/2023).

[57] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia. *Attention Is All You Need*. Dec. 5, 2017. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 02/12/2023).

[58] Wang, Meihong, Qiu, Linling, and Wang, Xiaoli. "A Survey on Knowledge Graph Embeddings for Link Prediction". In: *Symmetry* 13.3 (Mar. 2021). Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, p. 485. ISSN: 2073-8994. DOI: 10.3390/sym13030485. URL: https://www.mdpi.com/2073-8994/13/3/485 (visited on 02/18/2023).

[59] Wang, Weichuan, Xie, Zhiwen, Liu, Jin, Duan, Yucong, Huang, Bo, and Zhang, Junsheng. *MDistMult: A Multiple Scoring Functions Model for Link Prediction on Antiviral Drugs Knowledge Graph*. Nov. 29, 2021. arXiv: 2111.14480[cs]. URL: http://arxiv.org/abs/2111.14480 (visited on 02/28/2023).

[60] Wang, Zhen, Zhang, Jianwen, Feng, Jianlin, and Chen, Zheng. "Knowledge Graph Embedding by Translating on Hyperplanes". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 28.1 (June 21, 2014). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v28i1.8870. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8870 (visited on 01/23/2023).

[61] Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, and Yu, Philip S. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386. arXiv: 1901.00596[cs,stat]. URL: http://arxiv.org/abs/1901.00596 (visited on 01/24/2023).

[62] Xu, Keyulu, Hu, Weihua, Leskovec, Jure, and Jegelka, Stefanie. *How Powerful are Graph Neural Networks?* Feb. 22, 2019. arXiv: 1810.00826[cs,stat]. URL: http://arxiv.org/abs/1810.00826 (visited on 12/05/2022).

[63] Yang, Bishan, Yih, Wen-tau, He, Xiaodong, Gao, Jianfeng, and Deng, Li. *Embedding Entities and Relations for Learning and Inference in Knowledge Bases*. Aug. 29, 2015. arXiv: 1412.6575[cs]. URL: http://arxiv.org/abs/1412.6575 (visited on 12/05/2022).

[64] Ying, Rex, He, Ruining, Chen, Kaifeng, Eksombatchai, Pong, Hamilton, William L., and Leskovec, Jure. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. July 19, 2018, pp. 974–983. DOI: 10.1145/3219819.3219890. arXiv: 1806.01973[cs,stat]. URL: http://arxiv.org/abs/1806.01973 (visited on 12/05/2022).

[65] You, Jiaxuan, Gomes-Selman, Jonathan, Ying, Rex, and Leskovec, Jure. *Identity-aware Graph Neural Networks*. Feb. 5, 2021. DOI: 10.48550/arXiv.2101.10320. arXiv: 2101.10320[cs]. URL: http://arxiv.org/abs/2101.10320 (visited on 12/05/2022).

[66] Zhang, Chenxi, Peng, Xin, Sha, Chaofeng, Zhang, Ke, Fu, Zhenqing, Wu, Xiya, Lin, Qingwei, and Zhang, Dongmei. "DeepTraLog: trace-log combined microservice anomaly detection through graph-based deep learning". In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22: 44th International Conference on Software Engineering. Pittsburgh Pennsylvania: ACM, May 21, 2022, pp. 623–634. ISBN: 978-1-4503-9221-1. DOI: 10.1145/3510003.3510180. URL: https://dl.acm.org/doi/10.1145/3510003.3510180 (visited on 01/22/2023).

[67] ZHANG, SHUAI, Tay, Yi, Yao, Lina, and Liu, Qi. "Quaternion Knowledge Graph Embeddings". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/d961e9f236177d65d21100592edb0769-Abstract.html (visited on 01/24/2023).

[68] Zhang, Zhizhou, Ramanathan, Murali Krishna, Raj, Prithvi, Parwal, Abhishek, Sherwood, Timothy, and Chabbi, Milind. "CRISP: Critical Path Analysis of Large-Scale Microservice Architectures". In: ().

[69] Zhou, Tao, Lu, Linyuan, and Zhang, Yi-Cheng. "Predicting Missing Links via Local Information". In: *The European Physical Journal B* 71.4 (Oct. 2009), pp. 623–630. ISSN: 1434-6028, 1434-6036. DOI: 10.1140/EPJB/E2009-00335-8. arXiv: 0901.0553[physics]. URL: http://arxiv.org/abs/0901.0553 (visited on 02/04/2023).

[70] Zitnik, Marinka, Agrawal, Monica, and Leskovec, Jure. "Modeling polypharmacy side effects with graph convolutional networks". In: *Bioinformatics* 34.13 (July 1, 2018), pp. i457–i466. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty294. URL: https://doi.org/10.1093/bioinformatics/bty294 (visited on 12/05/2022).

List of Figures

List of Tables

Acronyms

GNNs Graph Neural Networks
GCN Graph Convolutional Network
ConvGNNs Convolutional Graph Neural Networks
RecGNNs Recurrent Graph Neural Networks
RecGNN Recurrent Graph Neural Network
CNN Convolutional Neural Network
NN Neural Network
NNs Neural Networks
WL Wesfeiler-Lehman
MLP Multi Layer Perceptron
ML Machine Learning
MRR Mean Reciprocal Rank
H@K Hits@K
KG Knowledge Graph
KGs Knowledge Graphs
CNNs Convolutional Neural Networks
UM Upstream Microservice
DM Downstream Microservice
RISE Research Institutes of Sweden
GCP Google Cloud Platform
NAISS National Academic Infrastructure for Supercomputing in Sweden
NSSA Self Adversarial Negative Sampling
RPC Remote Procedure Call
NLP Natural Language Processing

Acknowledgements

Lastly, I want to give a big thank you to all the people who supported me during this journey.

I want to thank my family, that has always tried to be close despite the distance from my hometown.

I want to thank my father and mother, who provided useful suggestions, and my sisters Clara and Giulia, who have always been happy to see me when I returned home.

A special thank you goes to my girlfriend, Ludovica, who has always supported me, despite the challenges of a long-distance relationship.

Moreover, I want to thank all my friends with whom I shared hours of study and preparation for the exams.

Thank you!