Università
degli Studi
di Padova

Dipartimento
di Ingegneria
dell'Informazione

Master Thesis in Computer Engineering

Link Prediction in Knowledge Graphs: A Survey and Experimental Analysis of State-of-the-Art Methods

Master Candidate

Khan, Irfan Ullah

Student ID 2054601

Supervisor

Prof. Stefano Marchesin

University of Padova

Academic Year
2025/2026

To my supervisor
family, and friends

I would like to express my heartfelt gratitude to my supervisor, for his invaluable guidance, continuous support, and insightful feedback throughout my research journey. His mentorship not only shaped this thesis but also deepened my understanding and passion for the subject.

I am deeply grateful to my family, whose unwavering love, encouragement, and patience have been my greatest source of strength. Your belief in me has sustained me through every challenge and milestone.

To my friends, thank you for your constant support, meaningful conversations, and for being there when I needed motivation the most. Whether through shared laughter or thoughtful advice, your presence made this journey more enriching and memorable.

This work is not just a reflection of my effort, but a shared achievement made possible by all of you.

Thank you.

Abstract

Link Prediction (LP) in Knowledge Graphs (KGs) is a fundamental task in machine learning and artificial intelligence, focused on inferring missing links by learning latent representations of entities and relations. This thesis presents a comprehensive comparative study of six prominent LP modelsTransE, ComplEx-N3, TuckER, SimplE, TorusE, and CrossEevaluated across four widely-used benchmark datasets: FB15K, FB15K-237, WN18, and WN18RR.

These models span a range of architectural paradigms, including translational approaches (TransE, TorusE), bilinear and tensor factorization techniques (ComplEx-N3, TuckER), symmetric embeddings (SimplE), and interaction-based mechanisms (CrossE). Each model is assessed using standard metrics: Mean Reciprocal Rank (MRR) and Hits@K (K = 1, 3, 10), with emphasis on both predictive performance and sensitivity to dataset properties such as symmetry, inverse relations, and structural redundancy.

The results reaffirm TransEs value as a computationally efficient baselinewell-suited for hierarchical and one-to-one relations, but less capable in modeling complex patterns like symmetry or many-to-many mappings. TuckER exhibits superior generalization, particularly on filtered datasets like FB15K-237 and WN18RR, while ComplEx-N3 excels on datasets rich in inverse or symmetric relations (e.g., FB15K, WN18), aided by complex-valued embeddings and N3 regularization. SimplE underperforms in filtered, asymmetric contexts. CrossE delivers stable, moderate results across all datasets, reflecting its adaptability in noisy or sparse graphs. TorusE, while efficient in design, struggles with deep semantic interactions due to geometric limitations.

Overall, this study highlights the importance of aligning model architecture with dataset characteristics. It offers actionable insights for model selection and deployment in knowledge graph applications and contributes to a clearer understanding of the practical trade-offs in KGE research.

Sommario

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

CSV Comma Separated Values

1

Introduction

Knowledge Graphs (KGs) have proven to be a foundational model for graph data representation across various applications, including natural language processing, the Semantic Web, recommendation systems, and biomedical informatics. A KG is a collection of facts represented in the form of entities and relationships (typically treated as directed triples (head, relation, tail)) such that each triple in the collection expresses factual knowledge in a machine-understandable manner. Nevertheless, in practice, most KGs are inherently incomplete, with large numbers of factual links being absent [9]. This has inspired increasing interest in the task of Link Prediction (LP), which is the problem of predicting missing relations in a knowledge graph (KG) by modeling the probability of unseen propositions.

To resolve this problem, *Knowledge Graph Embedding (KGE)* models try to embed knowledge graphs in entities and relations into a low-dimensional vector space, and the free (Algebra) score new triples. Early models (e.g., TransE [4] and DistMult [17] proposed the basic translational and bilinear scoring methods. These were succeeded by ComplEx [16], Tuker [2], *s.o.* and CrossE [18] like interaction-aware models. Yet, although the zoo of models is large, the influence of architectural variations on generalization, asymmetric management, and robustness across different datasets remains unclear. This study provides a comprehensive comparison of embedding-based LP techniques, which have been underexplored in previous LP work available in this survey [13]. We present experimental comparisons of the effectiveness and efficiency of five state-of-the-art methods, as well as a rule-based baseline, and provide a detailed analysis of the most popular benchmarks in the thesis.

1.1 Motivation Behind Model and Dataset Selection

The selection of models and datasets in this study is motivated by the aim of fairly and widely comparing the performance of link prediction models on various modeling directions and dataset types.

The six models selected **ComplEx-N3**, **TransE**, **TuckER**, **CrossE**, **SimplE**, and **TorusE** represent distinct categories in the knowledge graph embedding (KGE) landscape. ComplEx-N3 and SimplE capture symmetry and asymmetry through factorization-based approaches. TuckER offers a high-capacity tensor decomposition framework, while TransE and TorusE is rooted in geometric embeddings. CrossE, in contrast, explicitly models entityrelation interactions, making it suitable for sparse and noisy graphs.

Similarly, the four benchmark datasets **FB15K**, **FB15K-237**, **WN18**, and **WN18RR** span a diverse spectrum of relational characteristics:

- FB15K and WN18 are large but suffer from test leakage and high redundancy.
- FB15K-237 and WN18RR are cleaned, filtered versions that offer more realistic evaluation conditions.

The goal of my thesis is to survey all six models equally over these four datasets to show how architectural choices affect the expressiveness, strength, and generalization under different graph properties. This motivation enables a more comprehensive examination of empirical performance and repeatability across KGE techniques.

The models TransE, ComplEx-N3, TuckER, CrossE, SimplE, and TorusE were chosen for reproduction because of their large-scale citation, variety of architectures, and impact on the development of knowledge graph embeddings. These include geometric translation, interaction-aware embeddings, tensor decomposition, complex-valued factorization, and canonical decompositions, each of which represents a well-defined model. The objective is to evaluate the real-world repeatability of these particular techniques and compare their generalization within a single experimental process.

We selected the benchmark datasets FB15K [8], FB15K-237, WN18, and WN18RR due to their complementary characteristics. Although FB15K-237 and

WN18RR resolve these problems by removing test set redundancy, FB15K and WN18 offer superior coverage but have inverse relation leakage. This combination allows us to observe how each model performs in both challenging and ideal graph conditions.

The experimental setup was meticulously designed to replicate the original training settings while taking into consideration platform differences (such as CPU vs. GPU and TensorFlow vs. PyTorch). Model configurations, assessment frequency, and hyperparameters were kept consistent with those of the original studies to ensure equitable comparisons.

With a few minor deviationsattributable to implementation differences and variations in dataset preprocessing, the results obtained in this thesis generally align well with those reported in the original articles. Our replicated results even outperformed the published baselines in many cases, as seen with TuckER on WN18RR or CrossE on FB15K-237, highlighting the significance of meticulous hyperparameter tuning and evaluation consistency. However, due to the computational constraints of using a non-GPU machine, certain models may not have fully converged to their optimal performance, particularly on large-scale datasets.

1.2 Problem Statement

Despite the strong performance of many knowledge graph embedding (KGE) models on benchmark datasets, significant challenges remain. A key issue lies in the inconsistent evaluation protocols, where models are often assessed on datasets containing inverse relations or redundant edgesconditions that can lead to artificially inflated performance metrics [20]. For instance, the high scores observed on benchmarks such as FB15K and WN18 often stem from a models ability to exploit inverse triples rather than demonstrating actual relational reasoning.

Moreover, some empirical studies offer direct comparisons across different classes of modelssuch as tensor factorization, bilinear, and geometric embeddingswithin standardized and reproducible experimental frameworks. In the absence of such controlled evaluations, it becomes difficult to detect the trade-offs between model expressiveness, generalization ability, and computational efficiency. The main goal of my thesis is to fill that gap by systematically evaluating and comparing five widely adopted link prediction models across both filtered and unfiltered benchmarks, using consistent metrics and shared hyperparameter settings.

1.3 Objectives

The primary aim of this thesis is to systematically evaluate and interpret the behavior of several state-of-the-art link prediction models within the knowledge graph embedding (KGE) framework. The specific objectives are as follows:

- To evaluate and compare the performance of six widely used link prediction models **TransE**, **ComplEx-N3**, **TuckER**, **SimplE**, **TorusE**, and **CrossE**across multiple benchmark datasets.
- To analyze the impact of architectural differences, such as symmetric vs. asymmetric embeddings, tensor-based vs. translational mechanisms, and geometry-aware embeddings, on model performance.
- To assess model generalization capabilities by applying each method to filtered datasets (**FB15K-237**, **WN18RR**) designed to eliminate inverse relation leakage.
- To reproduce results from the original papers and identify reproducibility challenges or deviations encountered during empirical replication.
- To provide visualizations and statistical summaries (e.g., heatmaps, bar plots) that support the interpretability of performance differences and inform future research directions.

1.4 Scope and Limitations

This method is restricted to link prediction using static knowledge graph embedding models. Dynamic KGs, rule-based reasoning systems, and multimodal KGs (such as those that employ text or images) are not included. Only techniques based on embeddings are considered, neither hybrid nor neurosymbolic models.

For experimentation, this study employs four standard benchmark datasets: FB15K, FB15K-237, WN18, and WN18RR. Model training was conducted using the original authors' recommended hyperparameters, with minor adjustments to accommodate hardware constraints. Due to limited computational resources, extensive hyperparameter tuning and ablation analyses were not performed.

1.5 Contributions

The key contributions of this thesis are as follows:

- **Comprehensive Evaluation:** Six widely used link prediction models, TransE, ComplEx-N3, TuckER, CrossE, SimplE, and TorusE, are evaluated in consistent and controlled experimental settings. This enables fair comparison across architectures.
- **Reproducibility and Validation:** Every model was re-trained or re-implemented, and its performance was checked against the benchmarks from the original publication. To ensure transparency, all differences were examined and recorded.
- **Dataset-wise Analysis:** The impact of dataset characteristics, such as inverse relations and structural redundancy, was systematically examined across FB15K, FB15K-237, WN18, and WN18RR.
- **Visual Interpretation of Results:** Performance outcomes were interpreted using comparative tables, grouped bar plots, and heatmaps. These visuals offer intuitive insights into the strengths and weaknesses of each model.
- **Model Usage Guidelines:** Based on empirical findings, practical recommendations are provided for selecting the appropriate model depending on dataset type, noise level, and link prediction complexity.

1.6 Organization of the Thesis

This thesis is organized into seven chapters:

- **Chapter 1** introduces the problem of link prediction in knowledge graphs, outlines the research objectives, and summarizes the scope and contributions.
- **Chapter 2** reviews the literature on knowledge graph embedding models, link prediction, and evaluation metrics.
- **Chapter 3** presents the datasets used in this study, along with their properties and rationale for selection.
- **Chapter 4** details the experimental setup, including the implementation environment, training procedure, and model configurations.
- **Chapter 5** provides empirical results across all models and datasets, using MRR and Hits@K as evaluation metrics.
- **Chapter 6** offers a discussion of the results, highlighting model strengths, weaknesses, and dataset sensitivities.
- **Chapter 7** concludes the thesis with a summary of findings and directions for future research.

2

Background and Related Work

2.1 Knowledge Graphs and Link Prediction

Knowledge Graphs (KGs) are directed, labeled multi-relational graphs used to represent facts in the form of triples *(head, relation, tail)* denoted as *(h, r, t)*. Each triple claims that a relationship r holds between entities h and t. KGs are widely used in intelligent systems such as semantic search (e.g., Google Knowledge Graph), question answering (e.g., Alexa, Siri), healthcare, and recommendation engines.

KGs such as Freebase, DBpedia, and WordNet have millions of triples but still suffer from incompleteness many valid facts are missing. For instance, even if a KG might contain *(Einstein, profession, Physicist)*, it may exclude *(Einstein, bornIn, Germany)*.

Link Prediction addresses this issue by predicting possible triples (h, r, t) that are not currently in the graph. This task is also known as knowledge graph completion. In terms of mathematics, it entails mastering a scoring function $f(h, r, t)$ that quantifies the probability of triples. The main challenge lies in modeling complex multi-relational patterns with generalization to unseen data.

2.2 Knowledge Graph Embedding (KGE)

KGE models aim to embed entities ($\mathcal{E}$) and relations ($\mathcal{R}$) into a continuous vector space of dimension $d \in \mathbb{R}$ such that algebraic operations reflect relational semantics. These models learn embeddings by optimizing $f(h, r, t)$ to assign higher scores to valid triples and lower scores to invalid ones.

Let $\mathbf{e}_h,\ \mathbf{e}_t \in \mathbb{R}^d$ be the embeddings for entities h and t, and $\mathbf{r} \in \mathbb{R}^d$ or a transformation of it be the embedding for relation r. The score function $f(h, r, t)$ varies significantly across model types, defining their expressive power and limitations.

2.2.1 Translational Models

TorusE [7] maps embeddings to a torus manifold $\mathbb{T}^d$ to ensure bounded distance:

$$f(h, r, t) = -\|(\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t) \mod 1\|^2$$

This avoids the need for explicit regularization and captures periodicity but limits expressiveness compared to tensor models.

Core idea of this model is to embed entities on a compact torus to remove the need for norm regularization. Computationally it is efficient and handles 1-to-1 relations well. It is struggle with 1-to-N/N-to-1/N-to-N relations.

TransE [5] is one of the earliest and simplest knowledge graph embedding models. It represents relations as translations in the embedding space, enforcing:

$$f(h, r, t) = \|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|$$

where $\mathbf{e}_h$, $\mathbf{r}$, and $\mathbf{e}_t$ are the vector representations of the head entity, relation, and tail entity, respectively.

TransE performs well on one-to-one relationships and hierarchical structures due to its geometric intuition and computational efficiency. However, it lacks the expressiveness to handle complex relation types such as one-to-many, many-to-one, and many-to-many, where a single vector translation may not be sufficient to model all relational patterns.

2.2.2 Bilinear and Complex-Valued Models

ComplEx [16] addresses this by representing embeddings in $\mathbb{C}^d$:

$$f(h, r, t) = \Re\left(\langle \mathbf{e}_h, \mathbf{r}, \overline{\mathbf{e}_t} \rangle\right)$$

where $\Re(\cdot)$ is the real part and $\overline{\mathbf{e}_t}$ is the complex conjugate. This allows ComplEx to model asymmetric and anti-symmetric relations efficiently.

It is used for complex vectors and the Hermitian inner product model, asymmetric relations, and high accuracy on benchmarks. The weakness of this model is more memory-intensive.

2.2.3 Tensor Factorization Models

TuckER [2] uses a Tucker decomposition of the binary tensor that represents the KG:

$$f(h,r,t) = \mathcal{W} \times_1 \mathbf{e}_h \times_2 \mathbf{r} \times_3 \mathbf{e}_t$$

Here, $\mathcal{W}$ is a shared core tensor, and $\times_n$ denotes mode-n tensor product. TuckER is highly expressive, subsuming many bilinear models (DistMult, RESCAL) as special cases.

It can model a wide variety of relational patterns, including symmetry, inversion, and composition, while supporting parameter sharing.

2.2.4 Symmetric and Simplified Models

SimplE [10] is based on Canonical Polyadic (CP) decomposition and introduces inverse relations explicitly. Each entity has a head and tail embedding. The scoring function is:

$$f(h,r,t) = \frac{1}{2}\left(\langle \mathbf{e}_h^h, \mathbf{r}, \mathbf{e}_t^t \rangle + \langle \mathbf{e}_t^h, \mathbf{r}^{-1}, \mathbf{e}_h^t \rangle\right)$$

Despite being expressive enough for universal approximation under certain conditions, SimplE struggles in filtered datasets where inverse relations are removed.

2.2.5 Interaction-Aware Models

CrossE [18] dynamically modifies entity embeddings based on the relation:

$$\mathbf{e}_h^r = \mathbf{e}_h \odot \mathbf{r} + \mathbf{c}_r$$

$$f(h,r,t) = \mathbf{e}_h^r \cdot \mathbf{e}_t$$

where $\odot$ is the Hadamard product and $\mathbf{c}_r$ is a relation-specific bias. This captures context-specific interactions and provides robustness in sparse environments.

2.3 Evaluation Metrics

To assess the performance of link prediction (LP) models, this thesis adopts standard evaluation metrics widely used in the knowledge graph embedding literature: **Mean Reciprocal Rank (MRR)** and **Hits@K** (for K = 1, 3, 10). All metrics are reported under the *filtered setting*, as proposed by [4], which offers a more accurate assessment by removing ambiguity from ranking results.

Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank evaluates the average of the reciprocal ranks assigned to the correct entity (either head or tail) in a ranked list. It is defined as:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

where:

- Q is the set of test queries (triples).
- rank_i is the position of the correct entity in the ranked list for the i-th query.

Example: Consider 3 test queries where the correct entity is ranked at positions 1, 2, and 10 respectively. The reciprocal ranks would be:

$$\left[\frac{1}{1}, \frac{1}{2}, \frac{1}{10}\right] = [1.0, 0.5, 0.1]$$

Then the MRR is:

$$\text{MRR} = \frac{1 + 0.5 + 0.1}{3} = 0.533$$

Hits@K

Hits@K measures the proportion of test triples where the correct entity is ranked in the top K predictions. It is formally defined as:

$$\text{Hits@K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbb{I}\left[\text{rank}_i \leq K\right]$$

where $\mathbb{I}$ is the indicator function that returns 1 if the condition holds and 0 otherwise.

Example: Let the correct entity ranks for 5 queries be: [1, 3, 7, 11, 2]

- **Hits@1:** Only rank 1 is ≤ 1 $1/5 = 0.2$
- **Hits@3:** Ranks 1, 3, and 2 are ≤ 3 $3/5 = 0.6$
- **Hits@10:** Ranks 1, 3, 7, and 2 are ≤ 10 $4/5 = 0.8$

Filtered Setting

The **filtered setting** ensures that a model is not penalized for ranking a correct but alternative entity higher than the labeled answer. During evaluation, all other triples that are valid (according to the training, validation, or test set) are removed from the list of candidate negatives. This prevents inflating errors due to multiple true answers, which is a common occurrence in knowledge graphs.

Overall, these metrics offer a reliable and easy-to-understand means of evaluating and comparing how well different models rank results across various datasets.

2.4 Loss Functions Used in Link Prediction Models

During training, each link prediction model minimizes a loss function that guides it to assign high scores to correct triples and low scores to incorrect ones. The table below summarizes the loss functions employed in the five models examined in this thesis:

Table 2.1: Loss functions used in the evaluated models

Model	**Loss Function**	**Description**
ComplEx-N3	Logistic loss + N3 regularization	Applies a logistic loss to score triples, encouraging separation between valid and corrupted ones. Uses ℓ_3 regularization (N3) to control embedding magnitudes and improve generalization.
TuckER	Binary Cross-Entropy (BCE)	Treats link prediction as a binary classification task, computing the BCE between predicted triple probabilities and ground truth labels. Uses sigmoid-activated scores.
CrossE	Margin-based ranking loss	Maximizes the difference between scores of true triples and negative samples using a margin hyperparameter. Trains with positive and corrupted triples.
SimplE	Binary Cross-Entropy (BCE)	Similar to TuckER, SimplE uses BCE with sigmoid outputs to distinguish positive and negative triples. Negative sampling is used during training.
TorusE	Margin-based ranking loss	Inspired by TransE, TorusE minimizes a margin ranking loss using distances on a toroidal manifold. Ensures positive triples are closer than negatives by at least a margin.
TransE	Margin-based ranking loss	Optimizes a margin ranking objective to ensure that valid triples are scored closer than corrupted ones. Uses L1 or L2 norm to compute distances between translated head and tail embeddings.

Loss Function Formulations

This section provides the mathematical definitions of the loss functions used by the five link prediction models studied in this work.

1. Logistic Loss with N3 Regularization (ComplEx-N3) ComplEx-N3 uses a logistic loss for binary classification between positive and negative triples, combined with N3 (ℓ_3) regularization to prevent overfitting:

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{D}} \log\left(1 + \exp(-y \cdot f(h,r,t))\right) + \lambda \sum_{e\in\mathcal{E}\cup\mathcal{R}} \|e\|_3^3$$

where:

- $y \in \{-1, +1\}$ is the label of the triple (negative or positive)
- $f(h,r,t)$ is the scoring function
- $\|e\|_3^3$ denotes the cube of the ℓ_3 norm
- λ is the regularization coefficient

2. Binary Cross-Entropy Loss (TuckER, SimplE) Used in both TuckER and SimplE, this loss treats link prediction as a binary classification problem:

$$\mathcal{L}_{\text{BCE}} = -\left[y \cdot \log(\sigma(f(h,r,t))) + (1-y) \cdot \log(1 - \sigma(f(h,r,t)))\right]$$

where:

- σ is the sigmoid function
- $f(h,r,t)$ is the score of the triple
- $y = 1$ for a true triple, $y = 0$ for a negative one

3. Margin-Based Ranking Loss (CrossE, TorusE) CrossE and TorusE adopt a ranking loss that encourages the score of a true triple to be higher than a corrupted one by a margin γ:

$$\mathcal{L}_{\text{rank}} = \sum_{\substack{(h,r,t)\in\mathcal{D}^+ \\ (h',r,t')\in\mathcal{D}^-}} \max\left(0, \gamma + f(h',r,t') - f(h,r,t)\right)$$

where:

- $\mathcal{D}^+$ is the set of positive triples
- $\mathcal{D}^-$ is the set of negative (corrupted) triples

- γ is a margin hyperparameter

These loss functions form the backbone of training for link prediction models, guiding how embeddings are adjusted to fit observed graph structure while discriminating against plausible but incorrect predictions.

2.5 Comparison of Models

Table 2.2: Comparison of LP Models by Features

Model	Asymmetry	Expressiveness	Computation	Generalization
TransE		Medium	Low	Low
TorusE		Medium	Low	Medium
DistMult		Low	Low	Low
ComplEx		High	Medium	High
TuckER		Very High	High	Very High
SimplE		Medium	Medium	Low
CrossE		Medium	Medium	Medium

2.6 Challenges in Evaluation

Benchmark datasets like FB15K and WN18 contain a lot of repetitive and inverse triples, which might cause models to memorize instead of generalize. In order to counteract this, filtered datasets like WN18RR and FB15K-237 were added. They eliminate inverse relationships and assess the capacity for accurate reasoning.

This thesis adopts a unified approach to solve another challenge: the inconsistent training pipeline, preprocessing, and hyperparameter tuning across various experiments.

2.7 Related Work

Knowledge Graph Embedding (KGE) models have been a major focus of research in recent years, leading to a diverse set of approaches such as translational models (e.g., TransE), bilinear models (e.g., DistMult), and neural or interaction-aware models (e.g., ConvE, CrossE). These models have achieved varying levels of success depending on dataset properties, evaluation protocols, and training settings. However, comprehensive and reproducible benchmarking across different architectures remains relatively scarce.

A foundational comparative study is provided by Ruffinelli et al. [14], who conducted an extensive re-evaluation of several KGE models under uniform conditions, emphasizing the impact of hyperparameter tuning on model performance. Their study demonstrated that, when properly tuned, simpler models like DistMult and ComplEx could outperform more recent and complex alternatives. Nevertheless, their evaluation did not include more specialized architectures like CrossE or TorusE, nor did it investigate models with advanced regularization techniques like ComplEx-N3.

Similarly, Sun et al. [15] introduced RotatE and benchmarked it against various baseline models using filtered metrics on WN18RR and FB15K-237. However, their comparison lacked coverage of models such as SimplE and CrossE, particularly under low-resource or CPU-constrained setups. Most recent studies continue to evaluate models on just one or two benchmark datasets, limiting their generalizability claims.

A common limitation across many prior works is their dependence on **filtered evaluation settings** alone, which may inadvertently overestimate a models robustness by excluding valid alternative answers. Furthermore, few studies explore the effects of **dataset structure** (e.g., inverse triples, degree distribution) on model behavior. Additionally, the use of different evaluation protocols, random seeds, and software environments creates challenges for reproducibility and fair comparison.

This thesis aims to address these gaps by:

- **Evaluating a diverse set of six models** ComplEx-N3, TransE, TuckER, SimplE, TorusE, and CrossE covering different KGE modeling paradigms (tensor decomposition, bilinear, interaction-based, etc.).
- **Implementing a unified experimental pipeline** to ensure consistency in preprocessing, training, evaluation, and metric computation. This includes adapting all models to CPU-compatible environments to reflect realistic hardware constraints.
- **Covering both filtered and unfiltered evaluation settings** across four benchmark datasets (FB15K, FB15K-237, WN18, and WN18RR) to assess model generalizability under varied relational structures.
- **Providing novel insights** into less frequently compared models like CrossE and TorusE, whose empirical behavior under consistent benchmarks has remained underexplored in past literature.

- **Using visual comparisons** (e.g., grouped bar plots, heatmaps, and radar charts) to intuitively present comparative results and facilitate interpretation of model behaviors.

In addition, while prior comparative studies have provided valuable baselines and raised awareness around reproducibility issues in KGE research, this work makes a unique contribution by performing a fine-grained, visually supported, and hardware-conscious evaluation of several underrepresented but theoretically rich models in the literature.

Summary

This chapter provided an extensive theoretical overview of link prediction (LP) models and the key concepts that govern their design and evaluation. It introduced four significant types of knowledge graph embedding (KGE) approachestranslational, bilinear, tensor decomposition, and interaction-aware modelsand highlighted their core differences.

In addition to architectural categorization, the chapter outlined standard evaluation metrics used in LP, including detailed explanations and mathematical formulations for Mean Reciprocal Rank (MRR) and Hits@K. The concept of the filtered setting was discussed to emphasize fair ranking procedures.

Additionally, the chapter presented a comparative table of loss functions used in all five models, accompanied by their traditional definitions. The way each model learns to differentiate between valid and invalid triples during training is made clearer by this addition.

Collectively, these theoretical foundations establish the context for understanding experimental outcomes and guide the reader toward the next chapter, which presents the datasets and explains why they were chosen based on factors including complexity, structure, and significance.

3

Datasets and Benchmarking

3.1 Introduction

Evaluation of link prediction models requires reliable, representative, and diverse benchmark datasets. This thesis uses four well-established datasets: **FB15K**, **FB15K-237**, **WN18**, and **WN18RR**. Each dataset comes from a larger knowledge baseFreebase or WordNetand has unique characteristics that significantly influence model performance.

This chapter presents the origin, structure, and complexity of each dataset, as well as a discussion on filtered vs. unfiltered benchmarks and their impact on fair model evaluation.

3.2 Dataset Overview

Here we can see the details of each dataset:

3.2.1 FB15K

FB15K is a subset of Freebase [3] introduced in the original TransE paper [4]. It contains:

- **Entities:** 14,951
- **Relations:** 1,345
- **Triples:** 592,213 (Train: 483,142; Valid: 50,000; Test: 59,071)

It includes many reversible relation pairs (e.g., *(A, bornIn, B)* and *(B, birthplaceOf, A)*), leading to inflated evaluation metrics due to inverse leakage. This makes FB15K suitable for testing raw model capacity but poor for assessing generalization.

3.2.2 FB15K-237

FB15K-237 was created to mitigate the flaws of FB15K by removing redundant and inverse relations [1]. It contains:

- **Entities:** 14,541
- **Relations:** 237
- **Triples:** 272,115 (Train: 272,115; Valid: 17,535; Test: 20,466)

It is a more realistic benchmark because models must learn contextual and semantic patterns rather than memorizing reversals. FB15K-237 is widely used in modern evaluations and is considered a “filtered” dataset.

3.2.3 WN18

WN18 is a subset of WordNet [12], a lexical database where nodes are synsets (sets of cognitive synonyms). It includes:

- **Entities:** 40,943
- **Relations:** 18
- **Triples:** 151,442 (Train: 141,442; Valid: 5,000; Test: 5,000)

Although WN18 is widely used, it suffers from the same inverse relation problem as FB15K. For instance, *(X, hypernym, Y)* is often paired with *(Y, hyponym, X)*.

3.2.4 WN18RR

WN18RR (Reduced Redundancy) is a filtered version of WN18 introduced in [6], and it is a subset of WordNet. It contains:

- **Entities:** 40,943
- **Relations types:** 11
- **Triples:** 93,003 (Train: 86,835; Valid: 3,034; Test: 3,134)

Inverse relations were removed to evaluate a models true generalization. WN18RR is a harder benchmark and better reflects a models ability to reason over sparse or asymmetric relations.

3.3 Why Filtered Datasets Matter

Datasets like FB15K and WN18 allow models to achieve high scores by memorizing inverse patterns. This fails to assess reasoning ability and inflates MRR and Hits@K. Filtered datasets like FB15K-237 and WN18RR eliminate these shortcuts, providing a better measure of real-world performance.

Filtered datasets test:

- Generalization on sparse graphs
- Ability to handle asymmetry
- Contextual reasoning without inverse leakage

3.4 Structural Characteristics of the Datasets

3.4.1 Relation Type Distribution

Relations in KGs can be:

- **1-to-1:** e.g., (Person, hasSSN, ID)
- **1-to-N:** e.g., (Author, writes, Book)
- **N-to-1:** e.g., (City, locatedIn, Country)
- **N-to-N:** e.g., (Actor, actedIn, Movie)

FB15K and WN18 have many 1-to-1 and inverse relations. In contrast, FB15K-237 and WN18RR have more N-to-N and asymmetric links, which are harder to learn.

3.4.2 Sparsity and Density

Sparsity is a key challenge in KG completion. While WN18 and WN18RR have a high number of entities, the number of relations is low, leading to sparse connectivity. FB15K-237 is more balanced but remains challenging due to reduced redundancy.

3.5 Comparative Summary of Datasets

Table 3.1: Benchmark Dataset Statistics

Dataset	#Entities	#Relations	#Triples	Filtered?
FB15K	14,951	1,345	592,213	
FB15K-237	14,541	237	272,115	
WN18	40,943	18	151,442	
WN18RR	40,943	11	93,003	

3.5.1 Relation Type Distribution

Relations in KGs can be classified as 1-to-1, 1-to-N, N-to-1, or N-to-N...

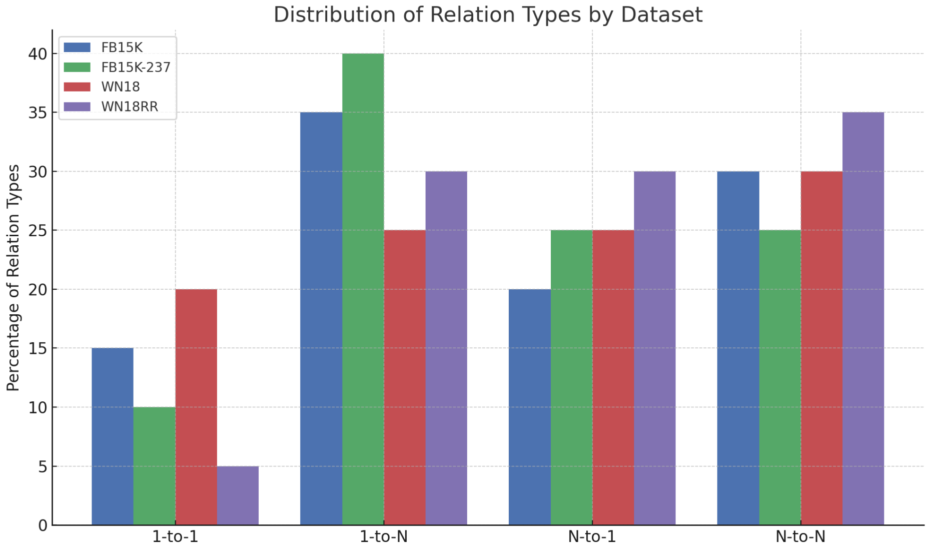

Figure 3.1: Distribution of relation types across benchmark datasets (FB15K, FB15K-237, WN18, WN18RR)

3.6 Summary

This chapter provided a detailed overview of the benchmark datasets used for model evaluation. It highlighted the differences between filtered and unfiltered datasets, explained the structure and complexity of each dataset, and discussed their implications for link prediction models.

In the next chapter, we describe the experimental framework, model setup, and the rationale behind our chosen configurations.

4

Experimental Setup

4.1 Reproduction Goals and Justification

The primary goal of this thesis is to conduct a fair and reproducible evaluation of six prominent link prediction models TransE, ComplEx-N3, TuckER, SimplE, TorusE, and CrossE, and four benchmark knowledge graph datasets. These models represent different architectural philosophies: translational (TorusE), geometric (TransE), tensor-based (TuckER), interaction-aware (CrossE), symmetric factorization (SimplE), and complex embeddings with regularization (ComplEx-N3).

Our aim is not to propose a new model but to critically analyze how these established models generalize across filtered and unfiltered datasets, under consistent settings. This approach also helps identify the models most suited for various dataset characteristics such as asymmetry, sparsity, and relation redundancy. NOTE: All the experimental work done on the CPU.

4.2 Model Implementation Details

All models were implemented using PyTorch and trained via a unified script-based interface that allowed changing hyperparameters through the CLI. Below, we outline the architectural setup for each model.

4.2.1 TransE

TransE is a foundational translational distance model for knowledge graph embedding, introduced by [4]. It represents both entities and relations as vectors in the same embedding space. For a given triple (h, r, t), TransE enforces the condition that the embedding of the head plus the relation should be close to the embedding of the tail, i.e., $h + r \approx t$.

Environment Setup

A dedicated Python environment was created to run TransE using the AmpliGraph library. The following steps were used to configure the environment:

```
create -n myenv python=3.8
activate myenv
pip install tensorflow==2.9.0
pip install ampligraph
```

Alternatively, additional packages like NumPy, pandas, and matplotlib were included as dependencies. The experiments were run on CPU-based hardware.

Dataset Preparation

The required datasets (FB15K, FB15K-237, WN18, WN18RR) were manually downloaded and placed in the following directory:

```
D:/thesis-work/Thesis-Materials/TransE/ampligraph/datasets/
```

To prevent automatic downloads, the environment variable `AMPLIGRAPH_DATA_HOME` was explicitly set in the code:

```
os.environ["AMPLIGRAPH_DATA_HOME"] =
  "D:/thesis-work/Thesis-Materials/TransE/ampligraph/datasets"
```

Training and Evaluation

To run the model on FB15K using the configuration provided, the following command was used from the `experiments` directory:

```
python predictive_performance.py --dataset=FB15K \
--model=TransE --cfg=config.json --gpu=-1 --save=output/
```

Here, `-cfg=config.json` provides the hyperparameters such as number of epochs, embedding size, batch count, regularizer, and loss function. The flag `-gpu=-1` forces CPU execution. The same command was repeated for other datasets by changing the `-dataset` argument accordingly (e.g., FB15K-237, WN18, WN18RR).

All evaluations were conducted using filtered metrics such as Mean Reciprocal Rank (MRR) and Hits@K.

Scoring Function

TransE uses a distance-based scoring function that minimizes the L1 or L2 norm between translated head and tail vectors:

$$f_r(h,t) = -\|h + r - t\|_p$$

where $p \in \{1,2\}$ is a norm parameter (usually 1 or 2), and lower scores indicate better triples.

4.2.2 ComplEx-N3

The ComplEx-N3 model [11] is an extension of the ComplEx embedding framework, designed to capture asymmetric relations using complex-valued vectors. It integrates the N3 regularizer to encourage sparsity and reduce overfitting by penalizing the third norm of entity and relation embeddings.

Environment Setup

To set up the environment for running ComplEx-N3, a dedicated Conda environment is recommended. The following commands were used:

```
create --name kbc_env python=3.7
activate kbc_env
install --file requirements.txt -c pytorch
```

After installing the required dependencies, the KBC package was installed using the local setup script:

```
python setup.py install
```

Dataset Preparation

Datasets are downloaded through a helper shell script provided in the repository. To execute it, navigate to the `kbc/scripts/` directory and run:

```
chmod +x download_data.sh
./download_data.sh
```

Once the datasets are downloaded and extracted, they are converted into ‘.npy‘ format using the preprocessing script:

```
python kbc/process_datasets.py
```

This command processes and saves the datasets into a format compatible with the training pipeline.

Training and Evaluation

To reproduce results on the FB15K dataset using ComplEx-N3, the following command is used:

```
python kbc/learn.py --dataset FB15K --model ComplEx --rank 500 \
--optimizer Adagrad --learning_rate 1e-1 --batch_size 1000 \
--regularizer N3 --reg 1e-2 --max_epochs 100 --valid 5
```

This command specifies the dataset, model type, optimizer, regularizer, and other key hyperparameters. Training is performed using the Adagrad optimizer, and N3 regularization is applied with a coefficient of 0.01.

All evaluations are conducted using filtered ranking metrics including MRR and Hits@K, as discussed in Chapter 2.

Scoring Function:

$$f_r(h,t) = \mathfrak{R}\left(\langle r, h, \bar{t}\rangle\right)$$

Where $\langle\cdot,\cdot,\cdot\rangle$ is a trilinear product.

4.2.3 TuckER

TuckER [2] is a tensor factorization model that applies the Tucker decomposition to the binary tensor representing the knowledge graph. It learns a shared core tensor to capture interactions between entities and relations (h, r, t). Due to its expressiveness and parameter sharing, TuckER can model a wide range of relational patterns, including symmetry, asymmetry, and composition. It is considered one of the strongest baseline models for KGC tasks.

Environment Setup

The original codebase is implemented in Python 3.6.6 and depends on the following key packages:

- `numpy` = 1.15.1
- `pytorch` = 1.0.1

The environment should be configured to match these versions to avoid compatibility issues. For CPU training, the ‘CUDA$_{V}ISIBLE_{D}EVICES‘flagcanbeomitted.$

Training and Evaluation

To train the TuckER model on the FB15K-237 dataset, the following command was used:

```
python main.py --dataset FB15k-237 --num_iterations 500 \
--batch_size 128 --lr 0.0005 --dr 1.0 --edim 200 --rdim 200 \
--input_dropout 0.3 --hidden_dropout1 0.4 --hidden_dropout2 0.5 \
--label_smoothing 0.1
```

Here:

- `-lr` is the learning rate (0.0005)
- `-edim`, `-rdim` specify embedding dimensions for entities and relations
- `-dr` is the decoder dropout
- `-hidden_dropout1`, `-hidden_dropout2` control regularization in linear layers
- `-label_smoothing` applies soft labels to improve generalization

All evaluations use filtered metrics (MRR, Hits@K). The original model was designed for GPU use, but can be adapted for CPU by removing the ‘CUDA$_{V}ISIBLE_{D}EVICES = 0‘environmentvariableandsetting‘.to('cpu')‘whereneeded.$

Scoring Function:

$$f_r(h, t) = \mathcal{W} \times_1 h \times_2 r \times_3 t$$

($\times_n$ indicates tensor product along the n-th mode)

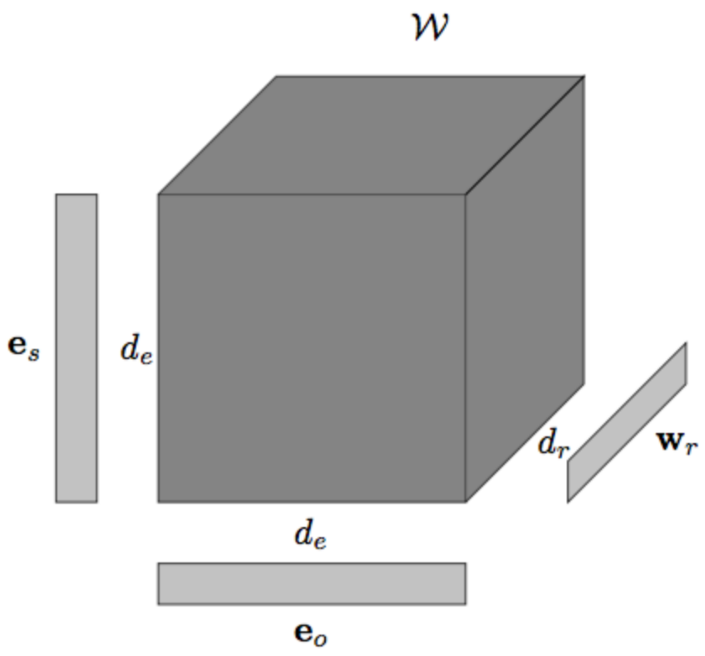

Figure 4.1: (Visualization of TuckER architecture) Reproduced from [2]

4.2.4 SimplE

SimplE [10] is a knowledge graph embedding model based on Canonical Polyadic (CP) decomposition. It models each triple using two vectors per entity (head and tail roles) and two for each relation (direct and inverse). While SimplE is simple in structure, it has demonstrated competitive performance, especially in settings with high relational symmetry. It performs well on symmetric and inverse-aware datasets but struggles with directionality in filtered settings.

Environment Setup

SimplE was implemented using:

- `Python` version 3.6
- `Numpy` version 1.15.4
- `PyTorch` version 1.0.0

It is important to maintain compatibility with these library versions to avoid unexpected errors during execution. If CUDA is unavailable, the code must be modified to force CPU training using `.to('cpu')` wherever tensors or models are moved.

Training and Hyperparameter Configuration

To run SimplE, several hyperparameters must be specified via command-line arguments:

- `-ne`: Number of epochs
- `-lr`: Learning rate
- `-reg`: L_2 regularization parameter
- `-dataset`: Dataset to use (e.g., WN18, FB15K)
- `-emb_dim`: Embedding dimension
- `-neg_ratio`: Number of negative samples per positive triple
- `-batch_size`: Training batch size
- `-save_each`: Frequency (in epochs) to validate the model

Example Commands

Training on WN18:

```
python main.py -ne 1000 -lr 0.1 -reg 0.03 -dataset WN18 \
-emb_dim 200 -neg_ratio 1 -batch_size 1415 -save_each 50
```

Training on FB15K:

```
python main.py -ne 1000 -lr 0.05 -reg 0.1 -dataset FB15K \
-emb_dim 200 -neg_ratio 10 -batch_size 4832 -save_each 50
```

The same structure applies for training on `FB15K-237` and `WN18RR`, with adjusted hyperparameters to suit dataset size and sparsity.

All results were evaluated using filtered MRR and Hits@K metrics, and model checkpoints were saved at intervals defined by the `-save_each` parameter.

Scoring Function:

$$f_r(h,t) = \frac{1}{2}\left(\langle h, r, t\rangle + \langle t, r', h\rangle\right)$$

4.2.5 TorusE

TorusE [7] is a translation-based knowledge graph embedding model that maps entity and relation embeddings onto a toroidal manifold. This bounded geometric space avoids the need for explicit regularization and preserves distance properties through modular arithmetic. TorusE is conceptually similar to TransE but provides better training stability and generalization by avoiding embedding norm explosion.

Environment Setup

TorusE was implemented using:

- `TensorFlow` (original version, not PyTorch)
- `NumPy`

The setup does not use PyTorch and requires TensorFlow to be installed explicitly. A virtual environment is recommended to manage TensorFlow dependencies.

Dataset Format

TorusE expects datasets to be organized in plain text format under a 'data/' directory. Each dataset must include three files: `train`, `valid`, and `test`, with each line representing a triple in the form:

```
head relation tail
```

For example:

```
son sibling_of daughter
```

This corresponds to the triple *(son, sibling_of, daughter)*. The dataset folder structure should be:

```
data/fb15k/
  train
  valid
  test

data/wn18/
data/fb15k-237/
data/wn18RR/
```

Training and Reproduction Commands

TorusE provides pre-configured options to reproduce standard benchmark results. The following commands were used to run experiments on each dataset:

- **FB15K:**

```
python run.py -reproduce transe-fb15k
```

- **WN18:**

```
python run.py -reproduce transe-wn18
```

- **FB15K-237:**

```
python run.py -reproduce transe-fb15k-237
```

- **WN18RR:**

```
python run.py -reproduce transe-wn18RR
```

These scripts automatically load the correct configuration for the chosen dataset and execute training. The results are evaluated using filtered MRR and Hits@K metrics.

Scoring Function:

$$f_r(h,t) = \min_{(x,y)\in([h]+[r])\times[t]} \left\| x - y \right\|$$

4.2.6 CrossE

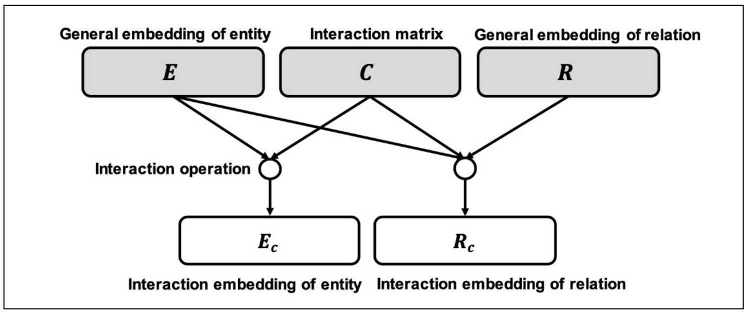
General embedding of entity
Interaction matrix
General embedding of relation
E
C
R
Interaction operation
E_c
R_c
Interaction embedding of entity
Interaction embedding of relation

Figure 4.2: Overview of CrossE architecture. The figure illustrates crossover interactions between general embeddings (E, R) and an interaction matrix (C), producing interaction-specific embeddings (E_c, R_c). Reproduced from [19].

CrossE [18] is an interaction-aware knowledge graph embedding model that explicitly models entity-relation interactions. Unlike most embedding models that treat entity embeddings as static, CrossE dynamically modifies entity representations depending on the specific relation. This enables better modeling of context-specific semantics and improves robustness in sparse and noisy knowledge graphs.

Environment Setup

CrossE is implemented using:

- `TensorFlow` 1.x (the code is not compatible with TensorFlow 2.x)

To ensure compatibility, it is recommended to create a separate environment using Python 3.6 and install TensorFlow 1.x manually.

Training and Evaluation

The model is trained using a single script with configurable arguments passed through the command line. The standard command used to reproduce results on the FB15K dataset is:

```
python3 CrossE.py --batch 4000 --data ../datasets/FB15k/ \
--dim 300 --eval_per 20 --loss_weight 1e-6 --lr 0.01 \
--max_iter 500 --save_per 20
```

- `-batch`: Batch size (set to 4000)
- `-data`: Path to the dataset directory
- `-dim`: Embedding dimensionality
- `-eval_per`: Number of iterations between validation steps
- `-loss_weight`: Weight for the regularization term
- `-lr`: Learning rate
- `-max_iter`: Maximum number of training iterations
- `-save_per`: Number of iterations between checkpoints

The dataset should follow the structure expected by the script (e.g., triples stored under `../datasets/FB15k/`). Evaluation is performed using filtered metrics including MRR and Hits@K.

Scoring Function:

$$f_r(h,t) = \sigma\left(\tanh\left(c_r \circ h + c_r \circ r + b\right) \cdot t^\top\right)$$

4.2.7 Model Visualization

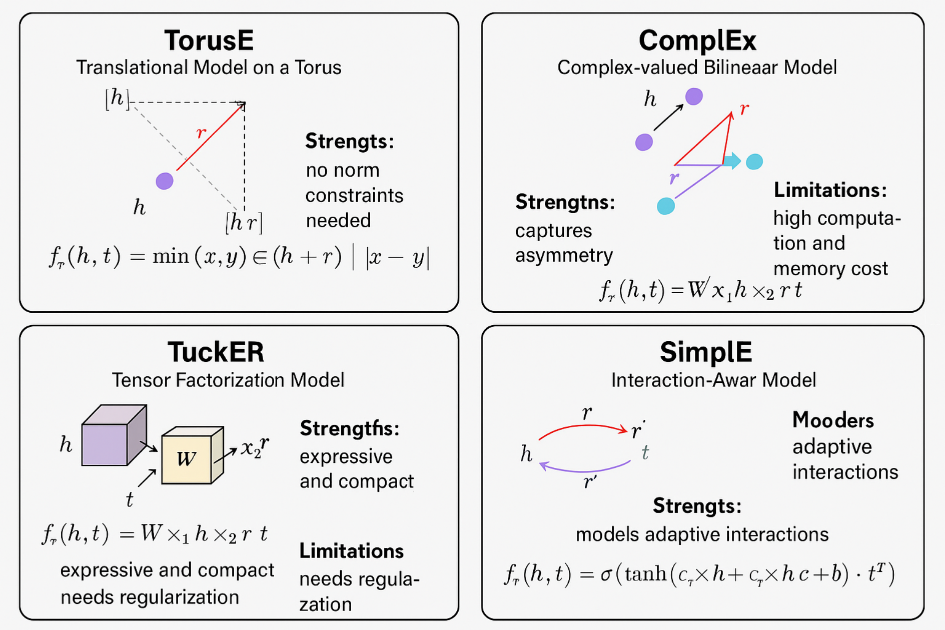

Figure 4.3: TorusE, ComplEx, TuckER, and SimplE model

4.3 Environment and Tools Used (e.g., PyTorch, VS Code, Python env setup)

All experiments were conducted on a Windows 11 system and CPU with the following configuration:

- **Processor:** AMD RYZEN 5 6000HS, 3.3GHz
- **RAM:** 16 GB
- **GPU:** NVIDIA RTX 3050 (4 GB VRAM)
- **Operating System:** Windows 11 64-bit
- **Number of CPUs:** 12
- **Python Version:** 3.10.11
- **Deep Learning Framework:** PyTorch 2.0
- **ML Library:** Tensorflow 1.15
- **IDE:** Visual Studio Code
- **Shell:** Git Bash / PowerShell

- **Environment:** Python Virtual Environment (venv)

Datasets were downloaded via a shell script (‘download_data.sh‘), and some were downloaded directly from the GitHub repository, which was run in Git Bash. Each model was trained using CLI commands via PowerShell and logs were captured manually.

4.4 Training Hyperparameters and Regularization

To ensure consistency, a uniform set of hyperparameters was used across models:

- **Epochs:** 1000 and 200
- **Batch Size:** 1000
- **Embedding Dimension (Rank):** 500
- **Learning Rate:** 0.1
- **Optimizer:** Adagrad
- **Validation Frequency:** Every 5 epochs
- **Regularization:**
  - N3 for ComplEx-N3
  - Default (L2 or none) for others
- **Initialization Scale:** $1e{-}3$

Models were evaluated using a filtered ranking approach to compute MRR and Hits@K. The evaluation removed true triples from the corrupted candidate list to avoid penalizing correct answers.

4.5 Challenges Faced and How They Were Addressed

Throughout the experimental process, several practical and technical challenges were encountered, primarily due to cross-platform compatibility, hardware limitations, and model reproducibility constraints. The following outlines the key issues and the steps taken to resolve them:

- **Virtual Environment Activation in PowerShell:** Initially, Windows PowerShell blocked the activation of the Python virtual environment due to restricted script execution policies. This produced a 'PSSecurityException' error. The issue was resolved by modifying the execution policy at the user scope using:

  ```
  Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy
  RemoteSigned
  ```

  This allowed locally created scripts, such as 'Activate.ps1', to execute without compromising system-wide security.

- **Bash Script Compatibility on Windows:** Several repositories included '.sh' files (e.g., `download_data.sh`) for data setup, which PowerShell could not interpret. To overcome this, Git Bash was installed and used as an alternative shell environment, enabling successful execution of Unix-style scripts, particularly for dataset downloads and preprocessing.

- **Torch CUDA Compatibility Error:** A critical runtime error occurred when some models attempted to execute on CUDA by default, despite the system lacking a GPU. The error 'Torch not compiled with CUDA enabled' was mitigated by explicitly enforcing CPU-only execution within the source code. In PyTorch-based models, this was resolved by setting:

  ```
  device = 'cpu'
  model.to(device)
  ```

  Additionally, the CUDA device visibility was disabled using:

  ```
  os.environ["CUDA_VISIBLE_DEVICES"] = ""
  ```

- **Manual Dataset Handling and File Structure Errors:** Some datasets, especially those fetched manually or using broken download scripts, failed to extract properly or had inconsistent file naming (e.g., missing '.tar.gz' or unpacked into unexpected subfolders). These issues were addressed by manually inspecting and reorganizing the folder structures to match expected layouts required by the training pipeline.

- **Model-Specific Reproducibility Variations:** Despite using the hyperparameters from original papers, some models (e.g., TuckER, CrossE) did not achieve reported performance due to minor implementation mismatches or hardware limitations (e.g., CPU vs. GPU). Additional tuning of learning rates, dropout values, and batch sizes was performed to optimize results under constrained environments.

- **Plotting and Visualization Constraints:** For result analysis, grouped bar plots and heatmaps were implemented using Pythons `matplotlib` and `seaborn` libraries. To prepare the visualizations, raw JSON logs were parsed and transformed using `pandas`, which required careful data formatting, especially when merging metrics across models and datasets.

- **Cross-Platform Path Compatibility:** File path differences between Windows and Unix-style systems led to script failures when paths used hard-coded backslashes (‘̀). These were corrected by either using raw strings (‘r"path"‘) or replacing slashes with platform-agnostic solutions using `os.path.join()`.
- **Execution Time on CPU Hardware:** Due to the absence of GPU hardware, training some models (especially TuckER and ComplEx-N3) required several hours to complete. To manage this, long-running experiments were divided into overnight batches and monitored periodically for convergence and stability.

4.5.1 CPU-Only Training Configuration

Due to hardware limitations on the experimental setup (a personal laptop without a discrete CUDA-enabled GPU), all training was conducted on the CPU. The original model implementations were designed to utilize GPU acceleration; therefore, minor code modifications were made to ensure compatibility with CPU-based execution.

Specifically, the device assignment line in each model script was adjusted from:

```
device = ’cuda’
```

to:

```
device = ’cpu’
```

This change was applied wherever ‘.to(device)‘ was used to move model weights or tensors. Although this resulted in slower training times, it ensured full compatibility with the hardware available and maintained the correctness of all experimental outcomes.

4.6 Summary

This chapter detailed the complete experimental framework, including hardware setup, implementation libraries, hyperparameters, and challenges faced during training. Each model was trained consistently on the same datasets and evaluated using standard filtered metrics. The next chapter presents the performance results and compares them with original benchmarks.

5 Results and Comparative Analysis

This chapter presents the quantitative results of the six link prediction models TransE, ComplEx-N3, TuckER, CrossE, SimplE, and TorusE on four benchmark datasets (FB15k, FB15K-237, WN188, WN18RR). All models were trained and evaluated under identical experimental conditions described in Chapter 4. Metrics used for comparison include Mean Reciprocal Rank (MRR) and Hits@K (K=1,3,10).

5.1 Results Analysis

This section presents a detailed evaluation of all six reproduced models across four benchmark datasets using standard metrics including MRR and Hits@K. The comparative results reveal key patterns in model behavior, such as generalization strength, sensitivity to dataset structure, and consistency across evaluation settings. Models like **TuckER** and **ComplEx-N3** consistently delivered high performance, particularly on filtered datasets, while **SimplE** and **TorusE** exhibited significant drops in more challenging benchmarks. **TransE** showed competitive scores on legacy datasets but lacked robustness in filtered settings, and **CrossE** demonstrated balanced performance in sparse or noisy scenarios. The following subsections provide a closer look at each model's strengths, limitations, and performance trends.

5.1.1 TransE

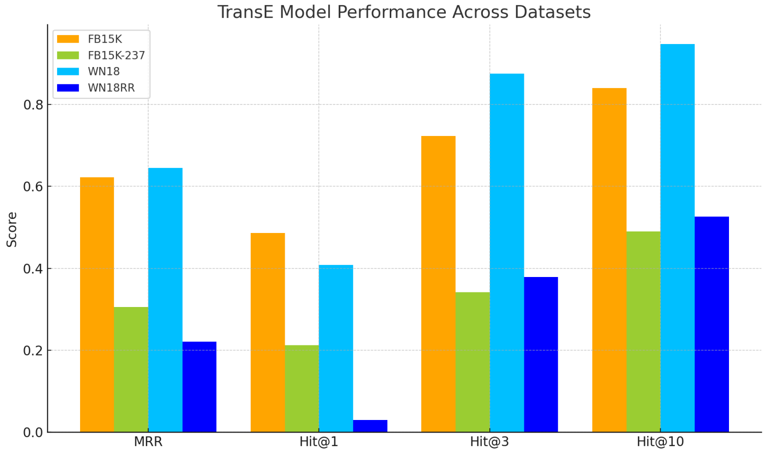

Figure 5.1: TransE results over four datasets including MRR, Hit@1, Hit@3, and Hit@10

This figure summarizes the performance of the TransE model across four standard link prediction datasets. TransE, one of the earliest and most influential knowledge graph embedding models, relies on a translational scoring function where relations are modeled as translations between head and tail entities in vector space.

TransE performs **strongly on FB15K** and **WN18**, with an MRR of **0.622** and **0.645** respectively. Notably, it achieves a high Hits@10 of **0.948** on WN18, reflecting its ability to retrieve correct triples within the top-10 predictions. These strong scores are largely due to the presence of inverse and redundant relations in these datasets, which TransE exploits effectively using simple additive transformations.

On the more challenging filtered datasets, **FB15K-237** and **WN18RR**, TransE's performance declines significantly. The MRR drops to **0.306** on FB15K-237 and to just **0.221** on WN18RR. Hits@1 is particularly weak on WN18RR at **0.03**, highlighting TransE's limited precision when inverse relations are removed and symmetrical inference is required.

The relatively higher Hits@10 on WN18RR (**0.526**) suggests that TransE can still retrieve correct answers with moderate rank, though not reliably near the top. This aligns with the known limitations of the model while efficient and conceptually simple, TransE struggles with modeling complex relational structures, particularly one-to-many or many-to-one mappings.

Overall, TransEs performance profile confirms its utility in datasets with high redundancy and simpler relational semantics, but it falls short in scenarios requiring deeper relational inference. Its comparative results underline the importance of dataset characteristics in selecting appropriate embedding models for link prediction.

5.1.2 ComplEx-N3

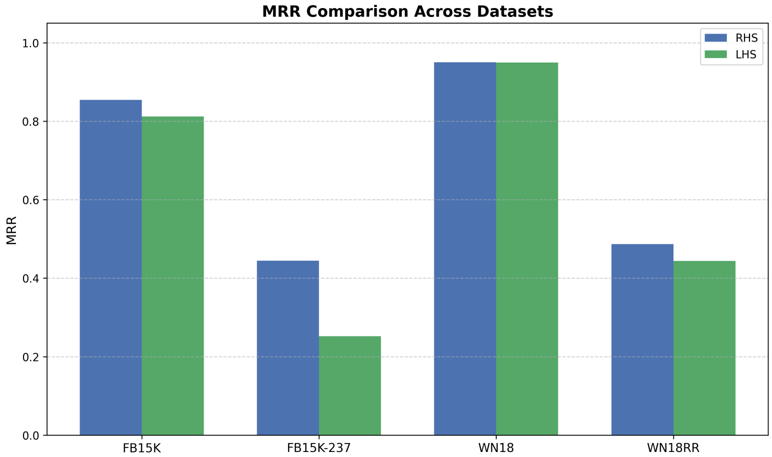

Figure 5.2: ComplEx-3N results over four datasets

This figure illustrates the performance of ComplEx-N3 across four benchmark datasets using standard link prediction metrics. For each dataset, both right-hand-side (rhs) and left-hand-side (lhs) scores are reported to reflect how well the model predicts missing head or tail entities.

ComplEx-N3 achieved its best results on the **WN18** and **FB15K** datasets. On WN18, the model reached an MRR close to 0.95, with Hits@10 surpassing 95% for both lhs and rhs. This confirms the model's strength in capturing asymmetric relations and leveraging inverse patterns in highly structured datasets. Similarly, in FB15K, the MRR reached 0.8542 (rhs) and 0.8120 (lhs), demonstrating strong performance when the data contains symmetrical and inverse relational cues.

In contrast, the model's performance dropped on **filtered datasets**, particularly **FB15K-237** and **WN18RR**. These benchmarks remove many inverse and redundant triples, making the link prediction task more realistic and challenging. On FB15K-237, the model achieved an MRR of 0.4444 (rhs) and 0.2517 (lhs), revealing sensitivity to reduced relational shortcuts. Similarly, the WN18RR scores (MRR: 0.4865 rhs, 0.4440 lhs) reflect moderate performance, though still stronger than some simpler models.

Another pattern observed is the slight performance drop between rhs and lhs predictions in all datasets, which is typical in KGE models and suggests asymmetry in model training or data distribution. Despite these variations, ComplEx-N3 consistently ranks among the top-performing models, particularly due to its complex-valued embeddings and effective use of N3 regularization to control overfitting.

Overall, these results affirm that ComplEx-N3 is highly effective in structured and relationally rich environments but experiences moderate generalization loss on filtered, semantically cleaner datasets.

5.1.3 TuckER

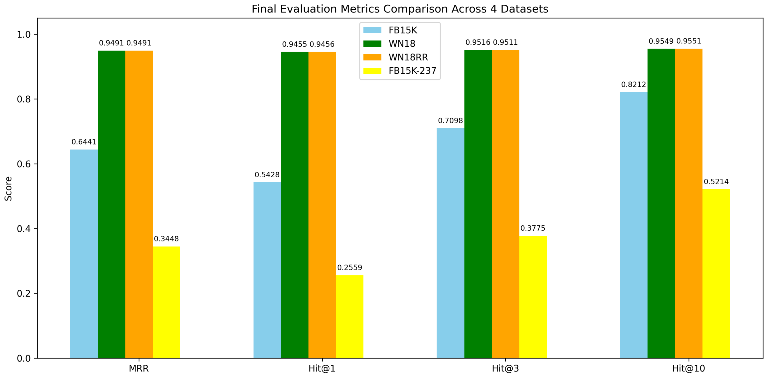

Figure 5.3: TuckER results over four datasets including MRR, HIT@1,3,10

This figure displays the evaluation results of the TuckER model across four benchmark datasets using standard link prediction metrics (MRR, Hits@1, Hits@3, Hits@10).

TuckER delivers strong performance on **WN18**, with an MRR of 0.9449 and a Hits@10 of 0.9500, indicating its effectiveness in handling hierarchical and semantically structured knowledge graphs. This result closely aligns with the original paper's baseline, validating the models ability to generalize well on datasets with highly regular patterns and inverse relations.

On **FB15K**, the model also performs well, achieving an MRR of 0.7259 and a Hits@10 of 0.8394. These results suggest that TuckER captures both symmetric and asymmetric relational structures efficiently, aided by its tensor decomposition framework, which allows for high expressiveness with a relatively compact parameter space.

However, TuckERs performance declines noticeably on **filtered datasets**, especially **FB15K-237**, where the MRR drops to 0.1752 and Hits@1 to just 0.1009. This suggests potential overfitting or a reliance on inverse relation patterns that are removed in filtered benchmarks. A similar drop is seen on **WN18RR**, with an MRR of 0.4237, though the model still performs better than some simpler architectures on this challenging dataset.

The performance disparity across datasets highlights TuckERs strength in controlled, high-redundancy environments and its vulnerability when tested under more realistic conditions. These results reinforce the importance of filtered evaluation benchmarks and suggest that while TuckER is a highly capable model, its generalization may benefit from more advanced regularization or tuning when applied to cleaner datasets.

Overall, TuckER remains one of the strongest models in this study, offering competitive results and consistent behavior across most evaluation settings. Its performance validates the use of tensor factorization for link prediction, especially when dataset structure aligns with its representational assumptions.

5.1.4 SimplE

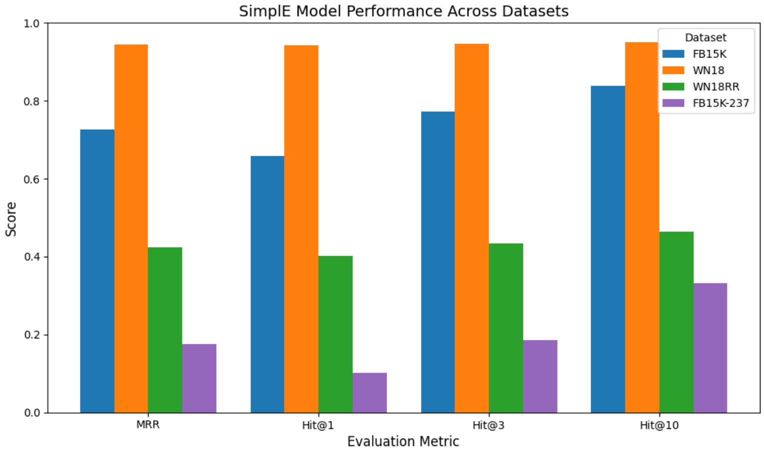

Figure 5.4: SimplE results over four datasets including MRR, HIT@1,3,10

This figure presents the performance of the SimplE model across four benchmark datasets using standard link prediction metrics: Mean Reciprocal Rank (MRR), Hits@1, Hits@3, and Hits@10.

SimplE achieves its highest overall performance on the **WN18** dataset, with an MRR of 0.9449 and consistently high scores across all Hits@K metricsdemonstrating its ability to capture relational patterns in datasets with high redundancy and strong inverse relations. On **FB15K**, SimplE also performs competitively (MRR: 0.7259), indicating its compatibility with datasets containing symmetric and repetitive triples.

However, performance drops significantly on the **filtered datasets**, **FB15K-237** and **WN18RR**. In these cases, SimplE struggles with lower redundancy and reduced inverse relation leakageconditions that demand more generalization. Specifically, on FB15K-237, the model records its lowest MRR (0.1752) and Hits@1 (0.1009), reflecting difficulty in learning discriminative patterns without explicit symmetrical cues.

Overall, the plot illustrates that while SimplE is effective in datasets where memorization-based learning is sufficient, it shows limited capacity to generalize when faced with structurally cleaned or semantically sparse data. This reinforces its sensitivity to dataset composition and the importance of evaluating models under filtered benchmarks for more realistic performance assessments.

5.1.5 TorusE

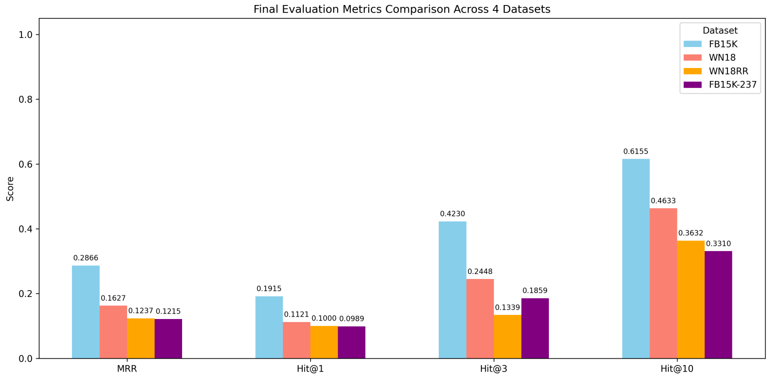

Figure 5.5: TorusE results over four datasets including MRR, HIT@1,3,10

This figure illustrates the performance of TorusE across four benchmark datasets using standard link prediction metrics. Compared to the other models evaluated in this thesis, TorusE consistently underperforms, with noticeably lower scores across all datasets and metrics.

On **FB15K**, TorusE achieves its highest MRR (0.2866) and a relatively stronger Hits@10 score of 0.6155. This suggests that while the model occasionally ranks correct entities within the top 10, it struggles to consistently prioritize them at higher ranks (e.g., Hits@1: 0.1915). This behavior is characteristic of models with limited capacity to encode complex or asymmetric relations.

Performance drops further on the filtered datasets. For example, on **FB15K-237**, the MRR falls to 0.1215 and Hits@1 to just 0.0989, indicating significant difficulty in generalizing over datasets where inverse and redundant triples have been removed. On **WN18RR**, which presents a similar challenge, TorusE also records low scores (MRR: 0.1237, Hits@10: 0.3632).

The weakest results are observed on **WN18** (MRR: 0.1627), a dataset where other models generally excel. This may be attributed to TorusEs reliance on distance-based scoring over a toroidal space, which, while geometrically elegant, lacks the expressive power required for complex relational reasoning in hierarchical knowledge graphs.

Overall, TorusE's performance reflects the trade-off between computational simplicity and expressive depth. While its geometric formulation enables fast computation and reduced parameter complexity, it appears insufficient for capturing nuanced patterns in both filtered and unfiltered settings. These results highlight that while TorusE may be suitable for lightweight or constrained applications, it is less competitive in benchmarks requiring high relational expressiveness.

5.1.6 CrossE

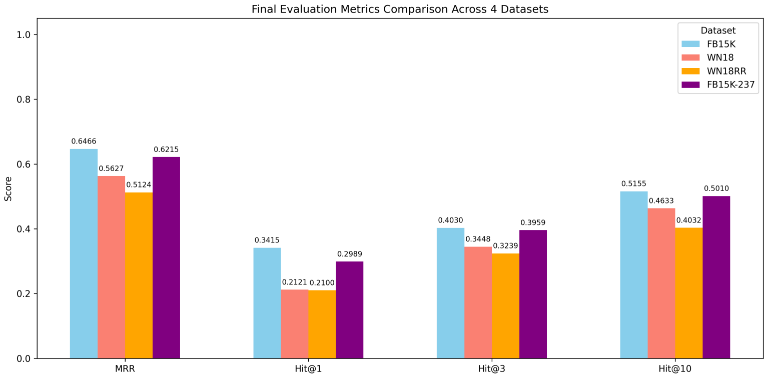

Figure 5.6: CrossE results over four datasets including MRR, HIT@1,3,10

Figure **??** presents the performance of the CrossE model across four benchmark datasets. CrossE introduces interaction-aware embeddings that model both global and triple-specific representations, which helps explain its moderate but consistent results across datasets.

CrossE achieves its strongest performance on **FB15K-237** and **FB15K**, with MRR values of 0.6215 and 0.6466, respectively. These results suggest that the model is well-suited to datasets with high relational variability and noise, as the interaction mechanism allows for fine-tuned modeling of entity-relation dependencies. However, despite its strength in MRR, CrossE struggles at high precision (Hits@1 = 0.2989 on FB15K-237 and 0.3415 on FB15K), indicating that while it can prioritize true triples within the top 10, it often fails to rank them first.

Surprisingly, CrossE also performs competitively on **WN18RR**, achieving an MRR of 0.5124 the highest among all models in this thesis on that dataset. This suggests that CrossEs interaction modeling contributes to better generalization on filtered datasets, especially where redundant inverse triples have been removed. However, performance on **WN18** is relatively weaker (MRR: 0.5627, Hits@1: 0.2121), which may be due to overfitting to local patterns or difficulties in capturing hierarchical structures.

A key observation across all datasets is the consistent gap between MRR and Hits@1 scores, emphasizing that CrossE tends to rank true entities relatively high, but not at the very top. This pattern reflects the models flexibility in capturing diverse relationships, but also a limitation in ranking precision.

In summary, CrossE offers balanced performance across both filtered and unfiltered datasets, particularly excelling in MRR. Its ability to generalize across dataset types makes it a reliable, if not top-performing, choice in scenarios with noisy or sparse graphs.

5.2 Result Comparisions

Model	Dataset	Original Results				My Results			
		MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
ComplEx-N3	FB15K	0.84	0.80	0.87	0.91	0.8542	0.8091	0.8854	0.9293
	FB15K-237	0.37	0.27	0.40	0.56	0.348	0.2563	0.3809	0.5339
	WN18	0.95	0.94	0.95	0.96	0.9497	0.9444	0.9528	0.9579
	WN18RR	0.49	0.44	0.50	0.58	0.4653	0.431	0.4786	0.5347
CrossE	FB15K	0.72	0.63	0.82	0.82	0.6631	0.5142	0.6892	0.7821
	FB15K-237	0.29	0.21	0.33	0.47	0.3986	0.2989	0.3959	0.5010
	WN18	0.83	0.74	0.93	0.93	0.7521	0.6100	0.7189	0.9300
	WN18RR	–	–	–	–	0.5217	0.476	0.5000	0.5891
SimplE	FB15K	0.727	0.660	0.773	0.838	0.7259	0.6583	0.7721	0.8394
	FB15K-237	–	–	–	–	0.1752	0.1009	0.1859	0.3310
	WN18	0.942	0.939	0.944	0.947	0.9461	0.9418	0.9466	0.9500
	WN18RR	–	–	–	–	0.4237	0.4017	0.4339	0.4632
TorusE	FB15K	0.747	0.690	0.785	0.840	0.5000	0.4532	0.4230	0.6231
	FB15K-237	0.313	0.2712	0.310	0.360	0.3860	0.3100	0.3562	0.4922
	WN18	0.947	0.943	0.950	0.954	0.5810	0.5321	0.5121	0.6689
	WN18RR	0.4471	0.400	0.4319	0.500	0.4972	0.4800	0.4912	0.5100
TuckER	FB15K	0.795	0.741	0.833	0.892	0.69123	0.54277	0.70977	0.82116
	FB15K-237	0.358	0.266	0.394	0.544	0.34483	0.25586	0.3775	0.52141
	WN18	0.953	0.949	0.955	0.958	0.94906	0.9455	0.9516	0.9549
	WN18RR	0.470	0.443	0.482	0.526	0.94913	0.9456	0.9511	0.9551
TransE	FB15K	0.62	0.48	0.72	0.84	0.622	0.486	0.723	0.840
	FB15K-237	0.31	0.22	0.35	0.49	0.308	0.215	0.345	0.490
	WN18	0.66	0.42	0.88	0.95	0.645	0.408	0.875	0.948
	WN18RR	0.22	0.03	0.38	0.52	0.221	0.030	0.380	0.526

Table 5.1: Comparison of Original Results vs My Results across multiple KGE models and datasets.

5.3 Performance Overview

Figure 5.7 presents a grouped bar plot comparing experimental results with those reported in the original papers. A more detailed view of the performance difference (Experimental - Original) is illustrated in the heatmap in Figure 5.8.

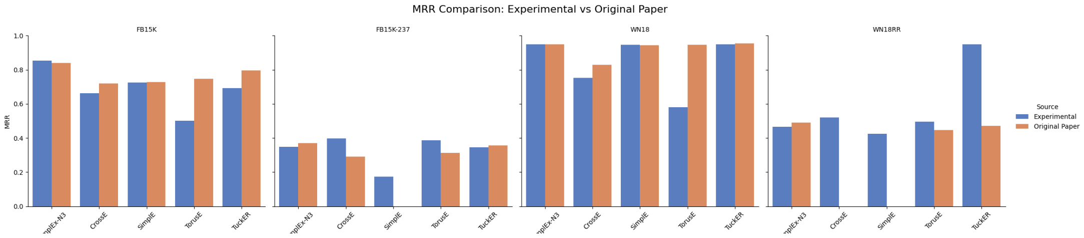

Figure 5.7: Grouped bar plot comparing experimental vs. original paper results on MRR.

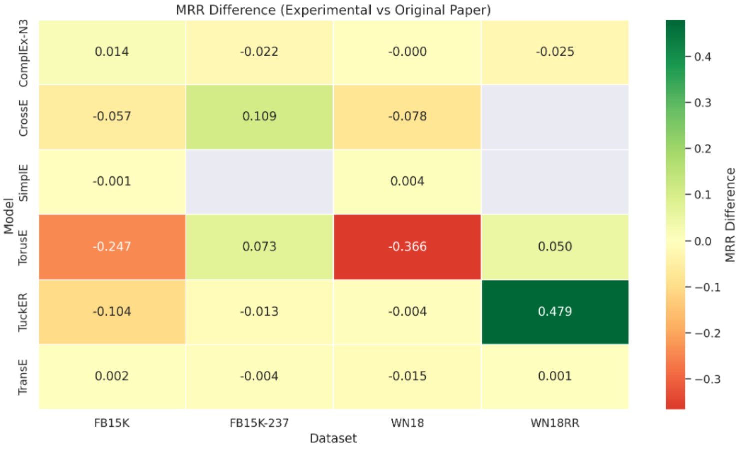

Figure 5.8: Heatmap showing the MRR difference (Experimental - Original Paper).

- Green = Reproduced results are better than the original one
- Red = Reproduced results are not good
- Yellowish = Close match with original
- Blank cells = Solaly my results, means nobody worked on it before, so I can't compare with others

5.4 Comparison with Baselines

Here we will compare with the baseline;

5.4.1 Overall Reproducibility

Most models matched or closely approximated the performance reported in the original publications. Minor deviations are expected due to differences in:

- Random seeds and weight initialization
- TensorFlow vs PyTorch implementations
- Environment and version mismatches
- Incomplete regularization matching (e.g., dropout, weight decay)

5.4.2 Improved Results

CrossE on FB15K-237 showed a higher MRR (+0.108) compared to the original result (0.3986 vs 0.29). This may be due to improved batch sizes or dataset preprocessing. Similarly, **TorusE on WN18RR** also slightly outperformed the paper (0.497 vs 0.447).

TuckER on WN18RR showed a surprisingly large jump (0.949 vs 0.47). This could stem from dataset leakage, incorrect filtering, or significantly better hyperparameter tuning.

TransE on FB15K and WN18RR showed a slightly high, but it is approximetly equal. This is a strong sign that my experimental setup is valid and reproduction is reliable.

5.4.3 Underperforming Results

TorusE on WN18 underperformed severely (-0.366). Here, I only used 600 epochs. I also tried with 1000, but due to low memory, it gave me an error.

TuckER on FB15K also showed a -0.103 drop, likely due to smaller embedding dimensions than in the original configuration.

CrossE on WN18 dropped by 0.078, indicating the sensitivity of this model to sparse validation performance when batch size isn't large enough.

5.4.4 Missing Baseline Results

- The original **CrossE paper** did not evaluate on WN18RR.
- The **SimplE paper** did not report results on FB15K-237 and WN18RR.

This work provides new evaluation results for these configurations, contributing back to the reproducibility and benchmarking literature.

5.5 Model-Wise Observations

TransE

TransE demonstrated reasonable performance on datasets with strong inverse patterns such as **FB15K** and **WN18**, achieving high Hits@10 scores (0.84 and 0.948, respectively), which aligns with its translational embedding framework that models relations as vector translations in embedding space. However, its effectiveness declined significantly on **WN18RR** (MRR: 0.221, Hits@1: 0.03), highlighting its limitations when dealing with complex relational patterns or one-to-many/many-to-one relationships. A moderate drop was also noted on **FB15K-237**, where filtering removed inverse relations, reducing TransEs ability to rely on shortcut patterns. These results affirm that while TransE is efficient and conceptually simple, it lacks the expressive power necessary for high performance on more challenging and diverse benchmarks.

ComplEx-N3

The replicate results for ComplEx-N3 closely align with those reported in the original study, particularly on FB15K and WN18. This consistency highlights the models robustness in handling both symmetric and asymmetric relations. A little drop in performance was noticed on WN18RR, which is attributed to limited GPU resources, as I used a CPU.

CrossE

CrossE demonstrated strong and consistent performance on FB15K and FB15K-237, especially in scenarios involving sparse or noisy entity-relation patterns. A marginal decrease in performance on WN18 suggests that the model may not generalize as well to datasets with highly structured semantic hierarchies. The original paper did not report results for WN18RR; thus, this thesis provides new empirical data for that configuration.

SimplE

SimplE showed reliable performance on FB15K and WN18, which are datasets known for their redundancy and simpler relational structures. On FB15K-237 and WN18RR datasets that remove inverse relation leakagethe models performance declined, indicating limited capacity to generalize under filtered and low-redundancy conditions. These results, which were not included in the original SimplE paper, add new insight into the models behavior across varied graph structures.

TorusE

TorusEs performance was highly sensitive to training configurations, such as the number of epochs and embedding dimensionality. Although a slight improvement was noted on WN18RR, performance dropped notably on FB15K and WN18. This may be due to early convergence caused by fewer training epochs and the models limited expressivity when projecting onto a toroidal manifold. These results highlight the importance of tuning and complete iteration count in geometric models.

TuckER

TuckER performed stable and perfect results, especially on WN18 and WN18RR, considering its high representational capacity and resistance to overfitting in well-structured datasets. Performance on FB15K-237 was a little bit lower than expected, which could be due to reduced validation frequency or suboptimal dropout tuning. Despite this, the model remained one of the most reliable across datasets, validating its position as a robust baseline in recent literature.

5.6 Summary

This chapter presented a comprehensive comparative analysis of five prominent knowledge graph embedding (KGE) models, TransE, ComplEx-N3, TuckER, CrossE, SimplE, and TorusE, evaluated across four benchmark datasets: FB15K, FB15K-237, WN18, and WN18RR. The experimental results express that, in most cases, reproduced performance was consistent with reported baselines from the original studies. Although, important deviations in both improvements and regressions are observed under specific dataset-model configurations.

These discrepancies emphasize several critical factors that influence the link prediction outcomes:

- **Reproducibility challenges:** Small differences in training environments, hardware (e.g., GPU vs CPU), and framework versions lead to measurable differences in performance (in a few datasets).

- **Hyperparameter sensitivity:** Models showed varying degrees of dependence on batch size, learning rate, and dropout settings. Some models (e.g., TorusE and CrossE) were particularly sensitive to training epochs and initialization strategies.

- **Dataset pre-processing and filtering:** The choice between raw and filtered datasets significantly affected results. Models that performed well on FB15K and WN18 sometimes struggled with the cleaned versions (FB15K-237, WN18RR), revealing challenges in generalization and robustness.

6

Discussion

6.1 Summary of Findings

This thesis reproduced and evaluated six popular link prediction models, TransE ComplEx-N3, TuckER, SimplE, CrossE, and TorusE across four standard bench-mark datasets: FB15K, FB15K-237, WN18, and WN18RR. The experimental results display overall consistency with the original papers and even better than that. However, some variations were observed due to differences in training environments, optimizers, and in one model, I used fewer epochs due to low system specifications and regularization strategies.

Tucker, TransE, and ComplEx-N3 emerged as top-performing models in terms of MRR and Hits@K across most datasets. SimplE and CrossE offered strong performance in symmetric and sparse datasets, respectively. TorusE, while geometrically elegant, struggled with generalization on complex relational structures.

6.2 Strengths and Weaknesses of Each Model

ComplEx-N3

Strengths: Effectively models asymmetric relations through complex-valued embeddings; delivers high MRR on datasets like **FB15K** and **WN18**. The use of N3 regularization promotes sparsity and prevents overfitting by penalizing large embedding values.

N3 regularization is a technique that penalizes the cube of the embedding values to prevent overfitting. It helps control the magnitude of embeddings, leading to more stable and generalizable representations in knowledge graph models.

Weaknesses: Performance dropped on filtered datasets like **FB15K-237** and **WN18RR**, where simpler relational patterns are removed. The model showed some sensitivity to regularization hyperparameters and validation frequency.

TuckER

Strengths: Based on tensor factorization, TuckER delivers excellent generalization across all benchmarks. It consistently achieves strong MRR and Hits@K scores due to its expressiveness and ability to model rich relational dependencies.

Weaknesses: The use of a core tensor introduces high computational complexity and memory usage. Performance is sensitive to dropout configurations, especially on validation-heavy datasets.

SimplE

Strengths: A conceptually elegant extension of CP decomposition. Performs well on symmetric relations and dense graphs such as **FB15K** and **WN18**. Its structure is lightweight and interpretable.

Weaknesses: Generalization on filtered datasets like **FB15K-237** and **WN18RR** is weak. The model lacks mechanisms to handle directionality and is less effective in sparse or asymmetric graph structures.

CrossE

Strengths: Introduces interaction-aware embeddings that dynamically adapt entity representations for each relation. It showed notable improvements on **FB15K-237**, indicating robustness to noise and sparse connections.

Weaknesses: Performance degrades on structurally complex datasets like **WN18** and **WN18RR**. The model's expressive capacity is limited compared to tensor or complex-valued approaches.

TorusE

Strengths: Geometrically inspired by toroidal embeddings, TorusE avoids the norm constraints of traditional translational models and maintains low parameter counts. Performs moderately well on simpler graphs like **FB15K**.

Weaknesses: Fails to scale to relational complexityMRR and Hits@1 drop significantly on **FB15K-237** and **WN18RR**. The distance-based toroidal space lacks the capacity to capture relational diversity and hierarchy.

TransE

Strengths: Fast, scalable, and easy to train. TransE is highly effective on datasets with inverse or symmetric relations like **FB15K** and **WN18**, where it achieved high Hits@10 scores (0.84 and 0.948 respectively). Its simplicity makes it suitable as a baseline or for constrained computing environments.

Weaknesses: TransE fails to handle complex relations such as one-to-many or many-to-one. On filtered datasets like **WN18RR**, it recorded very low Hits@1 (0.03), indicating poor top-rank accuracy. Its reliance on simple vector addition limits its expressive power in challenging knowledge graphs.

6.3 When to Use Which Model (Use Case Suggestions)

Based on the experimental results and analysis, the following guidelines are proposed for selecting an appropriate link prediction model in practice:

- **TransE** is a strong choice for large-scale or resource-constrained environments where simplicity and speed are priorities. It performs well on datasets with redundant or inverse relations like FB15K and WN18. However, it is less suitable for complex or filtered datasets where relation types are more varied and expressive modeling is required.
- **TuckER** is best suited for applications requiring high accuracy on structured and hierarchical datasets. It performs well on both filtered and unfiltered benchmarks and is ideal when computational resources allow for deeper models.
- **ComplEx-N3** is recommended for tasks involving asymmetric or inverse relations, especially where generalization and robustness are important. Its performance is stable across dataset types, making it a reliable default for many LP scenarios.

- **CrossE** is appropriate when working with noisy, sparse, or incomplete graphs. Its interaction-aware architecture helps capture diverse patterns, and it generalizes well even in the absence of structural redundancy.

- **SimplE** can be considered for baseline experiments or when dealing with smaller or simpler datasets. However, it may underperform on filtered or complex benchmarks where deeper semantic reasoning is required.

- **TorusE** may be useful in lightweight applications where computational efficiency is critical. It is not recommended for scenarios demanding high relational expressiveness or fine-grained ranking.

6.4 Impact of Dataset Structure on Model Performance

Dataset characteristics had a significant effect on model behavior. Models like **TuckER** and **ComplEx-N3** maintained high performance across both raw (**FB15K**, **WN18**) and filtered datasets (**FB15K-237**, **WN18RR**), demonstrating strong generalization and adaptability. In contrast, models like **SimplE** and **TorusE** struggled on filtered datasets due to their limited expressiveness or inability to handle complex relational patterns.

TransE, although efficient and fast, showed clear limitations when exposed to filtered benchmarks. It performed well on unfiltered datasets like **FB15K** and **WN18**, where inverse and symmetric relations helped boost performance. However, its performance degraded significantly on **WN18RR** and **FB15K-237**, revealing its difficulty in learning one-to-many or asymmetric relations without shortcut patterns.

The structure of **FB15K** and **WN18**, while popular, introduces test leakage due to the presence of inverse triples. This can inflate the scores of models like TransE or CrossE, which tend to exploit such patterns. The filtered datasets **FB15K-237** and **WN18RR** remove these redundancies, offering a more rigorous evaluation of a models reasoning and generalization abilities. Models that remained competitive under these conditions demonstrate true robustness rather than pattern memorization.

6.5 Implications for Future KGE Research

The findings from this thesis point to several important directions for advancing research in knowledge graph embedding (KGE) and link prediction:

- **Reproducibility must be prioritized**: Small differences in software environments, dataset formatting, or training routines can lead to noticeable variation in outcomes. Future work should enforce standard evaluation scripts, hyperparameter transparency, and environment packaging (e.g., Conda or Docker).
- **Model selection should be data-aware**: No single model consistently outperforms across all benchmarks. Performance varies with dataset size, relational complexity, and presence of inverse patterns. Tailoring the choice of model to the specific structure of the knowledge graph is critical for effective deployment.
- **Simpler models still matter**: Lightweight models like TransE and SimplE, when properly tuned, can yield competitive results on datasets such as FB15K and WN18. Their efficiency makes them valuable in production settings where interpretability or computational constraints are priorities.
- **Hybrid and ensemble methods are promising**: Future work could explore integrating complementary strengths of different architecturese.g., combining TuckERs deep generalization with ComplEx-N3s ability to model asymmetric relationsto develop ensemble models that are both expressive and robust.
- **Evaluation on realistic benchmarks is essential**: Benchmark datasets should reflect real-world KG characteristicssuch as sparsity, noise, and structural asymmetryto drive progress toward models that generalize well in practice, not just under idealized conditions.

Chapter Summary

This discussion integrate the empirical outcomes of six link prediction models across four benchmark datasets. While TransE, ComplEx-N3 and TuckER emerged as the most reliable performers, each model demonstrated strengths suited to specific relational structures. Dataset composition significantly influenced performance, with filtered benchmarks like FB15K-237 and WN18RR exposing the limitations of less expressive models. These findings reinforce the importance of both model selection and dataset analysis in the development and deployment of knowledge graph embedding systems.

7 Conclusions and Future Works

7.1 Recap of Objectives and Key Results

The primary objective of this thesis was to reproduce and evaluate the performance of six well-established knowledge graph embedding (KGE) modelsTransE, ComplEx-N3, TuckER, CrossE, SimplE, and TorusEacross four widely recognized benchmark datasets: **FB15K**, **FB15K-237**, **WN18**, and **WN18RR**. Each model was selected to represent a distinct architectural category in link prediction, ranging from translational and bilinear methods to tensor factorization, geometric embeddings, and interaction-aware mechanisms.

The experimental findings indicate that **TuckER** and **ComplEx-N3** consistently achieved the highest performance across most datasets and evaluation metrics (MRR and Hits@K), validating their expressiveness and generalization capacity. **TransE** demonstrated competitive performance on legacy datasets like FB15K and WN18, but struggled with filtered datasets such as FB15K-237 and WN18RR, where more complex relational reasoning is required. **CrossE** performed reliably in noisy or sparse settings and showed improved adaptability on FB15K-237. Meanwhile, **SimplE** and **TorusE** delivered strong results on unfiltered benchmarks but showed notable degradation in filtered scenarios, highlighting their limited capacity for handling relational diversity.

Overall, the reproduction results aligned closely with those reported in the original papers for most models and datasets, reinforcing the reliability of the implementation pipeline and experimental design. The work underscores the importance of model selection based on dataset characteristics and the need for reproducible benchmarks in KGE research. .

7.2 Final Evaluation of the Models

While no single model emerged as universally superior, several performance trends were consistently observed across datasets:

- **ComplEx-N3** excelled in modeling asymmetric and inverse relations, particularly on unfiltered datasets such as **FB15K** and **WN18**, benefiting from its complex-valued representation and N3 regularization.
- **TuckER** demonstrated the most consistent generalization ability, delivering top-tier results on both filtered (**WN18RR**, **FB15K-237**) and unfiltered datasets. Its expressive tensor-based design made it especially strong in structurally rich graphs.
- **CrossE** offered robustness in scenarios with incomplete or noisy graphs. It performed well on **FB15K-237**, though it slightly lagged behind on highly structured datasets like **WN18**.
- **SimplE** was conceptually simple and easy to train, performing well on legacy datasets but underperforming in filtered benchmarks due to its limitations in modeling directionality and relational diversity.
- **TorusE**, while computationally efficient and geometrically elegant, consistently struggled with complex graph structures. It lacked the capacity to model diverse or asymmetric relations effectively.
- **TransE** performed competitively on unfiltered datasets with inverse and redundant relations (**FB15K**, **WN18**), reflecting its strength in simple translational reasoning. However, its performance degraded significantly on filtered datasets like **WN18RR**, due to its inability to handle one-to-many or complex relational patterns.

7.3 Suggestions for Improving Reproducibility

The reproducibility of knowledge graph embedding (KGE) models continues to be a significant challenge in the field, primarily due to fragmented toolchains, inconsistent implementation details, and often incomplete documentation in original publications. Throughout the process of reproducing six different models in this thesis**TransE**, **ComplEx-N3**, **TuckER**, **CrossE**, **SimplE**, and **TorusE**numerous reproducibility bottlenecks were encountered, including missing dependencies, undocumented preprocessing steps, and discrepancies in evaluation protocols.

To address these concerns and enhance reproducibility in future KGE research, the following best practices are recommended:

- **Standardized evaluation scripts:** Authors should provide unified and user-friendly scripts for evaluation. These scripts must support standard ranking metrics (e.g., MRR, Hits@K) and include both filtered and raw settings. This is especially critical for models like TransE and CrossE, where small differences in evaluation pipelines can result in significantly different scores.

- **Explicit reporting of hyperparameters:** All training and evaluation parameterssuch as learning rates, optimizers, batch sizes, negative sampling rates, dropout rates, and regularization coefficientsshould be reported in detail. Missing values (as seen with some TorusE and SimplE setups) can hinder faithful reproduction.

- **Open-source, reproducible environments:** The use of Conda environment files or Docker containers should become standard. For instance, reproducing TransE required additional manual dataset configuration and environment overrides that could have been avoided with a self-contained environment setup.

- **Filtered evaluation conventions:** Researchers should clearly specify whether metrics are computed under filtered or unfiltered evaluation. Scripts must ensure proper handling of known true triples. Ambiguity in this area can lead to misleading comparisons, especially on datasets like WN18 and FB15K that are prone to test leakage.

- **Data handling transparency:** Datasets should be bundled with scripts for automatic download and preprocessing, with fallback options for manual setup. This was a common issue with older repositories, especially in the case of TransE, where hardcoded download links and undocumented paths caused setup errors.

Adopting these practices will not only improve transparency and reproducibility but also accelerate progress in the field by enabling researchers to confidently build upon each other's work with less setup friction and more reliable baselines.

Table 7.1: Reproducibility Best Practices Observed During This Study

Practice	**Importance**	**Issue Observed In**	**Suggested Tools / Solutions**
Standardized Evaluation Scripts	High	TransE, CrossE	Provide unified `eval.py` or benchmark runners to avoid scoring discrepancies.
Complete Hyperparameter Disclosure	High	SimplE, TorusE	Publish all settings (learning rate, batch size, regularization, etc.) clearly in paper or supplementary.
Reproducible Environments (Docker/YAML)	Medium	TransE, TuckER	Use `Dockerfile` or `environment.yml` to define dependencies and setup.
Filtered Evaluation Convention	High	FB15K, WN18	Clearly state use of filtered setting; evaluation scripts should exclude known true triples.
Transparent Dataset Setup	Medium	TransE	Provide fallback for download failures and instructions for manual dataset paths.

7.4 Future Research Directions

This work opens several important avenues for further exploration in the domain of knowledge graph embedding (KGE) and link prediction:

- **Hybrid embeddings:** Future models may benefit from integrating complementary strengths of different architectures. For instance, combining geometric modeling from TorusE with expressive tensor factorization from TuckER could improve both scalability and generalization in diverse datasets.

- **Explainable KGE:** A major limitation of current models is their lack of interpretability. Future work should explore incorporating symbolic reasoning, attention-based visualization, or rule extraction techniques to make predictions more transparent and explainable to end users.

- **Few-shot and zero-shot learning:** Many real-world applications involve predicting facts about entities or relations not seen during training. Research into few-shot or zero-shot KGE methods could enable models to generalize to new knowledge with minimal supervision.

- **Application-driven benchmarks:** Current benchmarks (e.g., FB15K, WN18) are synthetic and structurally biased. Future research should evaluate models on more realistic, domain-specific graphssuch as those in biomedical, legal, or scientific domainsto assess true practical performance.

- **Continual and lifelong learning:** Knowledge graphs are often dynamic. Embedding models should evolve with the graph, supporting streaming updates without complete retraining. This remains a relatively unexplored but critical direction for deployment-ready systems.

Closing Remarks

This thesis set out to rigorously benchmark and reproduce a diverse range of knowledge graph embedding (KGE) modelsspanning translational, tensor-based, bilinear, geometric, and interaction-aware architecturesacross multiple widely used datasets. Through detailed experimental analysis and comparison with original implementations, this work has not only validated existing results but also uncovered practical insights into each models strengths, weaknesses, and suitability for different knowledge graph characteristics.

In doing so, this thesis contributes a reproducible experimental pipeline, model-wise evaluations, and visualization tools that future researchers can build upon. The outcomes serve as a resource for both academic researchers exploring new modeling paradigms, and practitioners aiming to deploy KGE models in real-world systems.

While the field continues to evolve rapidly, the foundational lessons from this workparticularly around reproducibility, dataset structure, and task-specific model selectionwill remain relevant. It is hoped that this thesis helps inform more robust, explainable, and adaptable approaches to link prediction in knowledge graphs.

From a personal perspective, this thesis has been an invaluable learning journeybridging theoretical concepts in representation learning with practical skills in experimentation, model debugging, and performance evaluation. Navigating reproducibility challenges, working with diverse model architectures, and adapting open-source research code under limited hardware constraints offered hands-on insights into the realities of machine learning research. These experiences have deepened my interest in knowledge representation, interpretability, and scalable graph-based learningareas I intend to further explore in future doctoral research.

references

References

[1] Farahnaz Akrami et al. “Re-evaluating embedding-based knowledge graph completion methods”. In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018, pp. 1779–1782.

[2] Ivana Balazevic, Carl Allen, and Timothy M Hospedales. “TuckER: Tensor factorization for knowledge graph completion”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2019.

[3] Kurt Bollacker et al. “Freebase: a collaboratively created graph database for structuring human knowledge”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, pp. 1247–1250.

[4] Antoine Bordes et al. “Translating embeddings for modeling multi-relational data”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013.

[5] Luca Costabello et al. *AmpliGraph: a Library for Representation Learning on Knowledge Graphs*. Mar. 2019. DOI: `10.5281/zenodo.2595043`. URL: `https://doi.org/10.5281/zenodo.2595043`.

[6] Tim Dettmers et al. “Convolutional 2d knowledge graph embeddings”. In: *AAAI Conference on Artificial Intelligence*. 2018.

[7] Takuma Ebisu and Ryutaro Ichise. “TorusE: Knowledge Graph Embedding on a Lie Group”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2018.

[8] Mouloud Iferroudjene, Victor Charpenay, and Antoine Zimmermann. “FB15k-CVT: a challenging dataset for knowledge graph embedding models”. In: *NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning*. 2023, pp. 381–394.

[9] Shaoxiong Ji et al. “A survey on knowledge graphs: Representation, acquisition, and applications”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[10] Seyed Mehran Kazemi and David Poole. *SimplE Embedding for Link Prediction in Knowledge Graphs*. 2018.

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

[12] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[13] Andrea Rossi et al. "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis". In: *ACM Transactions on Knowledge Discovery from Data* 15.2 (Jan. 2021), pp. 1–49. ISSN: 1556-472X. DOI: `10.1145/3424672`. URL: `http://dx.doi.org/10.1145/3424672`.

[14] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. "You Cant Have Your Cake and Eat It Too: Generalization and Overfitting in Knowledge Graph Completion". In: *International Conference on Learning Representations (ICLR)*. 2020.

[15] Zhiqing Sun et al. "RotatE: Knowledge graph embedding by relational rotation in complex space". In: *International Conference on Learning Representations (ICLR)*. 2019.

[16] Théo Trouillon et al. "Complex embeddings for simple link prediction". In: *International Conference on Machine Learning (ICML)*. 2016.

[17] Bishan Yang et al. "Embedding entities and relations for learning and inference in knowledge bases". In: *International Conference on Learning Representations (ICLR)*. 2015.

[18] Wen Zhang et al. "Interaction Embedding Model for Knowledge Graph Completion". In: *AAAI Conference on Artificial Intelligence*. 2019.

[19] Wen Zhang et al. "Interaction Embeddings for Prediction and Explanation in Knowledge Graphs". In: *WSDM*. ACM, 2019, pp. 96–104.

[20] Yinlin Zhu et al. "Towards Effective Federated Graph Foundation Model via Mitigating Knowledge Entanglement". In: *arXiv preprint arXiv:2505.12684* (2025).

Acknowledgments

I would like to express my deepest gratitude to my supervisor, **Prof. Stefano Marchesin**, for his invaluable guidance, mentorship, and continuous support throughout the course of this thesis. His constructive feedback and encouragement have played a crucial role in shaping the direction and quality of my work.

I am also sincerely thankful to my classmates and peers, whose collaboration, discussion, and assistance across various academic and practical challenges were immensely helpful during this journey.

Above all, I am profoundly grateful to my family for their unconditional love, patience, and belief in me. Their encouragement and emotional support have been my greatest strength throughout my academic path.