Neural Methods for Link Prediction in Knowledge Graphs

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Adrian Kochsiek

Mannheim, 2025

Dekan	Prof. Dr. Claus Hertling, Universität Mannheim
Referent	Prof. Dr. Rainer Gemulla, Universität Mannheim
Korreferent	Prof. Dr. Simone Paolo Ponzetto, Universität Mannheim
Korreferent	Prof. Dr. Gerard de Melo, Hasso Plattner Institute

Tag der mündlichen Prüfung: 15.01.2025

Abstract

Integrating neural models with multi-relational data poses a significant challenge, mainly due to the complexity and highly interconnected nature of such data, which is prevalent in domains such as social networks, knowledge graphs, and biomedical databases. In this context, this thesis explores three key challenges: effectively capturing multi-relational data for efficient reasoning and integration into downstream applications, integrating structural and textual information, and managing the evolving nature of multi-relational graphs. Knowledge graph embeddings (KGE) offer a promising approach, learning low-dimensional representations for entities and relations to enhance tasks such as recommendation, question answering, and visual relationship detection. However, scalability and quality assurance in KGE models remain issues, especially with large-scale graphs. Recent frameworks address scalability through parallelization techniques, but their impact on model quality requires further study. While extensive hyperparameter optimization can considerably improve the resulting embedding quality, current optimization methods do not scale to large-scale graphs. Moreover, integrating textual information with structural data in a simple, scalable, and versatile manner remains an open problem.

Addressing these challenges, this thesis makes several key contributions. First, it evaluates and improves the efficiency and effectiveness of parallel training techniques for KGE models. By re-implementing and extensively testing various parallelization methods, the study identifies optimal techniques for large-scale KGE training, demonstrating significant speedups while maintaining model quality. Second, it proposes GraSH, a scalable hyperparameter optimization algorithm based on successive halving, achieving state-of-the-art results on large-scale knowledge graphs with minimal search budgets. Third, the thesis introduces KGT5-context, a novel approach that integrates structural and textual information by posing link prediction as a sequence-to-sequence task, achieving or surpassing state-of-the-art results in link prediction and downstream tasks. Finally, it presents the Wikidata5M-SI benchmark for evaluating model performance on dynamic graphs with emerging entities in a realistic, large-scale setting.

Zusammenfassung

Die Integration von neuronalen Modellen mit multi-relationalen Daten stellt eine bedeutende Herausforderung dar, aufgrund der Komplexität der verbundenen Daten, wie sie in Bereichen wie sozialen Netzwerken, Wissensgraphen und biomedizinischen Datenbanken vorkommen. In diesem Kontext untersucht diese Arbeit drei zentrale Herausforderungen: (i) das effektive Erfassen von multi-relationalen Daten für effiziente Inferenz und die Integration in nachgelagerte Anwendungen, (ii) die Integration von strukturellen und textuellen Informationen sowie (iii) die Handhabung der dynamischen Eigenschaft von multi-relationalen Graphen. Knowledge Graph Embeddings[a] (KGE) bieten einen vielversprechenden Ansatz, indem sie niedrigdimensionale Repräsentationen für Entitäten und Relationen lernen, um Aufgaben wie Empfehlung, Fragebeantwortung und visuelle Beziehungserkennung zu verbessern. Allerdings bleiben Skalierbarkeit und Qualitätssicherung bei KGE-Modellen Herausforderungen. Neue Frameworks adressieren die Skalierbarkeit durch Parallelisierungstechniken, aber deren Auswirkungen auf die Modellqualität bedürfen weiterer Untersuchung. Während umfangreiche Hyperparameter-Optimierung die Qualität der resultierenden Embeddings erheblich verbessern kann, skalieren aktuelle Optimierungsmethoden nicht auf große Graphen. Zudem bleibt die Integration von textuellen Informationen mit strukturellen Daten auf einfache, skalierbare und flexible Weise ein ungelöstes Problem.

Zur Bewältigung dieser Herausforderungen leistet diese Arbeit mehrere wesentliche Beiträge. Erstens bewertet und verbessert sie die Effizienz und Effektivität paralleler Trainingstechniken für KGE-Modelle. Durch die Neuimplementierung und umfangreiche Prüfung verschiedener Parallelisierungsmethoden identifiziert die Studie optimale Techniken für das Training von KGE Modellen auf großen Graphen und zeigt eine signifikante Reduzierung der Trainingszeit bei gleichzeitiger Erhaltung der Modellqualität. Zweitens wird GraSH eingeführt, ein skalierbarer Hyperparameter-Optimierungsalgorithmus basierend auf “Successive Halving”, der mit minimalen Suchbudget Ergebnisse auf dem Stand der Technik auf großen Wissensgraphen erzielt.

[a] Mögliche Übersetzung des Fachbegriffs ins Deutsche: Wissensgrapheinbettungen.

Drittens führt die Arbeit KGT5-Kontext ein, einen neuartigen Ansatz, der strukturelle und textuelle Informationen integriert, indem er die Vorhersage von neuen Verbindungen im Graphen als Sequenz-zu-Sequenz-Aufgabe darstellt und dabei den derzeitigen Stand der Technik in der Linkvorhersage und darauf aufbauenden Aufgaben erreicht oder übertrifft. Schließlich wird die Wikidata5M-SI-Benchmark vorgestellt, um die Modellleistung auf großen, dynamischen Graphen mit neuen Entitäten in einem realistischen Setting zu evaluieren.

Contents

Acknowledgments

I never initially planned to pursue a PhD. With limited interest in research, I started my master's thesis primarily as a means to earn a degree. However, during those six months, my advisor, Rainer Gemulla, along with his PhD students at the time, Daniel Ruffinelli and Samuel Broscheit, showed me how fascinating and enjoyable research could be. I want to thank all three of you for sparking my interest in research. Thank you, Rainer Gemulla, for guiding me on this journey, for pointing me toward exciting research directions, and especially for your rigorous, detailed feedback, which greatly improved my research, writing, and presentations. This thesis would not have been possible without your support.

I am also grateful to Alexander Renz-Wieland for engaging discussions on parallel training methods for embeddings and for the support at the start of my PhD. Additionally, I extend my gratitude to Apoorv Saxena for a great collaboration and for steering my research into the NLP domain. I would like to express my sincere thanks to the DWS group for their unwavering support along the way. In particular, I want to highlight Andreea Iana and Patrick Betz for making every little break more enjoyable.

This work would not have been possible without the abundance of support I received from my family and friends during my studies and throughout my PhD. Thank you all! Finally, I am especially thankful to my incredible fiancée, Linsha Li, for being my greatest source of support and encouragement over the years, and for all the wonderful moments we've shared and will continue to share. Having you in my life makes everything better!

Chapter 1

Introduction

"I am convinced that the crux of the problem of learning is recognizing relationships and being able to use them."

Christopher Strachey in a letter to Alan Turing, 1954

The integration of neural models with multi-relational data represents a pivotal challenge. Multi-relational data, which encompasses a myriad of interconnected entities and their relationships, is omnipresent in numerous domains such as social networks (El-Kishky et al. 2022), knowledge graphs (Suchanek et al. 2007; Vrandečić and Krötzsch 2014), and biomedical databases (Gene Ontology Consortium 2004; Robinson et al. 2008). These complex data structures provide rich contextual information that, if effectively harnessed, can significantly enhance the performance and applicability of neural models. This thesis addresses three key questions within this context.

(i) How can multi-relational data be effectively captured for efficient reasoning within the graph and integration into downstream applications?

(ii) How can this structural information be integrated with textual information describing the entities and relations in such multi-relational graphs?

(iii) How can the ever-changing nature of multi-relational graphs be managed when capturing and integrating its information?

A prominent modeling approach to addressing the first objective is knowledge graph embeddings (KGE). These neural models learn low-dimensional representations, termed embeddings, for each entity and relation within a knowledge graph

(KG) or any other multi-relational graph. KGE models are typically trained on the task of link prediction, i.e., predicting new relationships between the entities in the graph. The resulting embeddings, in turn, allow neural models to integrate the relational knowledge stored in these representations to be used as features for downstream tasks, such as recommendation (El-Kishky et al. 2022; Iana et al. 2022), question answering (Huang et al. 2019; Saxena et al. 2020, 2022), query approximation (Hamilton et al. 2018), as well as visual relationship detection (Baier et al. 2017). While KGs can scale to millions or even billions of entities and contain a vast set of facts, much of the literature on KGEs focuses on small KGs (Ali et al. 2021a; Bordes et al. 2013; Chen et al. 2021; Ruffinelli et al. 2020). For large-scale KGs and integration into real-world applications, it is crucial that KGEs are scalable and ensure high embedding quality effectively capturing multi-relational information.

To address scalability, recently, frameworks capable of training KGE models for large-scale KGs by parallelization across multiple GPUs or machines have been proposed (Lerer et al. 2019; Zhu et al. 2019; Zheng et al. 2020, 2024). These frameworks employ various parallelization techniques, enabling the handling of large-scale KGs with reasonable training times. However, the impact of these methods on model quality has not been comprehensively studied. Ensuring high embedding quality, Ali et al. (2021a) and Ruffinelli et al. (2020) explored the impact of hyperparameter choices and training techniques for KGE models on the resulting model quality. They found the search space to be vast and hyperparameter choices to be dataset- and model-dependent. For instance, the best configuration found for one model may perform poorly with another. But these studies focused on small KGs only. Conducting such an extensive hyperparameter search is generally not cost-efficient or even feasible on large-scale KGs, where KGE training is expensive in terms of runtime, memory consumption, and storage cost. To mitigate these excessive costs, the prior studies focussing on scale either forgo hyperparameter optimization (HPO) altogether or reduce runtime and memory consumption by employing various heuristics (Lerer et al. 2019; Zhang et al. 2022; Zheng et al. 2020). The former approach leads to suboptimal quality, while the latter's impact on quality and cost has not been studied in a principled way. While KGEs are a valuable approach addressing research objective (i) on a small scale, the described challenges on efficient and effective training and tuning need to be addressed for the use of KGEs on large-scale data.

Next to valuable structural information, KGs often provide useful textual information, such as entity mentions and descriptions. In addition to the described scaling challenges, KGEs typically disregard these valuable textual sources. However, integrating both structural and textual information is desirable and should be achieved

through a simple and scalable architecture that maintains high quality and allows for versatile downstream task integration. Initial integration approaches either initialize or concatenate KGE embeddings with pretrained text embeddings (Hu et al. 2021; Daza et al. 2021; Xie et al. 2016). More sophisticated approaches directly finetune Transformer-based architectures on graph tasks (Yao et al. 2019; Clouatre et al. 2021; Wang et al. 2022). But, all of these approaches lack at least one of the integration goals of simplicity, scalability, quality, and versatility. A sufficient answer to research objective (ii) should fulfill all four goals.

Finally, even if a model proves as a solution to the first two research objectives, for a useful and long-term integration into real-world applications, such a model needs to be able to handle the ever-changing nature of the underlying information source. For example, new users joining social networks, new products being released, or new events occurring. Many models can only answer queries for such emerging entities after retraining, which becomes prohibitively expensive for large-scale graphs. Consequently, multiple benchmarks (Albooyeh et al. 2020; Daza et al. 2021; Galkin et al. 2021; Shah et al. 2019; Wang et al. 2019) evaluate models on link prediction queries involving emerging entities. However, emerging entities may come with varying amounts of information, including contextual facts and textual details. Benchmarks should allow for a fine-grained evaluation on large-scale graphs while considering the varying amounts of information. This level of detail and scale is not provided in a realistic setting by existing benchmarks.

This thesis addresses these challenges by evaluating and improving existing parallel training techniques for KGE models in terms of efficiency and effectiveness, proposing a scalable hyperparameter optimization approach for large-scale KGEs, successfully integrating structural and textual information through a text-based sequence-to-sequence link prediction approach, and finally evaluating model performance on evolving KGs with new entities joining the graph.

Contributions

Efficient and effective training for large-scale KGE models. To evaluate the efficiency and effectiveness of available parallelization methods for KGE training, we re-implemented large-scale training techniques proposed by prior work (Lerer et al. 2019; Zheng et al. 2020; Zhu et al. 2019) within a common framework and conducted an extensive experimental study. We discovered that prior evaluation methodologies are often inconsistent and can be misleading, as degradations in model quality due to parallel training may go undetected. Our results indicate that current (combinations

of) parallel training methods tend to have a negative impact on embedding quality or fail to provide substantial speedups. However, we found that efficient and effective parallel training of large-scale KGE models is achievable with a careful selection of techniques, which is dependent on the dataset. For example, training a large-scale KGE model for the full Freebase KG on 8 GPUs achieved a 7× speedup and model quality competitive with sequential methods, surpassing prior results.

Hyperparameter optimization for large-scale KGE models. To allow for high embedding quality of large-scale KGEs and address the lack of efficient HPO approaches, we explored the effective use of a given HPO budget to obtain high-quality KGE models. To do so, we first summarized and analyzed both the cost and quality of various approximation techniques proposed by prior work (Lerer et al. 2019; Tu et al. 2019; Zhang et al. 2022; Zheng et al. 2020). Building on these findings, we developed GraSH, an efficient multi-stage HPO algorithm for large-scale KGE models based on the popular successive halving algorithm (Jamieson and Talwalkar 2016). Our extensive experimental study demonstrated that GraSH achieves state-of-the-art results on large-scale KGs with a low overall search budget, equivalent to only three complete training runs.

Integration of textual and structural information. To integrate textual and structural information directly, we framed KG link prediction as a sequence-to-sequence task and trained an encoder-decoder Transformer model (Vaswani et al. 2017) on this task. This simple yet powerful approach, which we call *KGT5*, achieved scalability by using compositional entity representations and autoregressive decoding for inference. It allows to further simplify the learning problem by integrating a graph lookup before inference. This extension, termed *KGT5-context*, achieved or outperformed state-of-the-art results on two large-scale link prediction benchmarks and demonstrated versatile application in downstream tasks such as question answering.

Evaluation of model performance on emerging entities. To evaluate model performance on emerging entities in a realistic, large-scale setting, we introduced the *Wikidata5M-SI* benchmark. This benchmark enables evaluation of model performance on queries involving emerging entities with varying amounts of contextual facts and textual information on a large-scale KG. Our experimental study with recent link prediction approaches revealed that performance for emerging entities significantly lags behind that for seen long-tail entities. There is generally a trade-off between performance on seen and emerging entities, and the proper integration of contextual and textual information warrants further exploration.

Outline

Before delving into the details of the contributions, we introduce relevant fundamentals in Chapter 2. This includes an introduction to knowledge graphs and the task of link prediction, a categorization of link prediction models, as well as a description of common training and evaluation approaches. Chapter 3 provides an overview of parallel training approaches for KGE models and presents an extensive experimental study evaluating their efficiency and effectiveness, and finally suggests improvements. In Chapter 4, we present an efficient and scalable hyperparameter optimization approach for KGE models on large-scale graphs. Subsequently, in Chapter 5, we address the integration of textual and structural information by treating link prediction as a sequence-to-sequence task. In Chapter 6, we focus on the dynamic nature of multi-relational graphs and introduce a new benchmark for link prediction that includes previously unseen entities and report on an experimental study comparing existing link prediction approaches. Finally, we summarize our findings and conclusions in Chapter 7.

Publications

The work presented in this thesis is based on the following publications.

- Kochsiek, A., & Gemulla, R. (2021).
  Parallel training of knowledge graph embedding models: a comparison of techniques.
  In Proceedings of the VLDB Endowment, 15(3).
- Kochsiek, A., Niesel, F., & Gemulla, R. (2022).
  Start small, think big: On hyperparameter optimization for large-scale knowledge graph embeddings.
  In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD).
- Saxena, A., Kochsiek, A., & Gemulla, R. (2022).
  Sequence-to-Sequence Knowledge Graph Completion and Question Answering.
  In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL, Volume 1: Long Papers).
- Kochsiek, A., Saxena, A., Nair, I., & Gemulla, R. (2023).
  Friendly Neighbors: Contextualized Sequence-to-Sequence Link Prediction.

In Proceedings of the 8th Workshop on Representation Learning for NLP (Repl4NLP@ACL).

- Kochsiek, A., & Gemulla, R. (2023).
  A Benchmark for Semi-Inductive Link Prediction in Knowledge Graphs.
  In Findings of the Association for Computational Linguistics (EMNLP).

Further, this thesis builds upon the following publication.

- Broscheit, S., Ruffinelli, D., Kochsiek, A., Betz, P., & Gemulla, R. (2020).
  LibKGE-A knowledge graph embedding library for reproducible research.
  In Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations (EMNLP).

Chapter 2

Fundamentals

"All knowledge is connected to all other knowledge. The fun is in making the connections."

Arthur C. Aufderheide, n.d.

In this chapter, we introduce fundamental concepts and methods used throughout this thesis. We begin by presenting the concept of a knowledge graph and the task of link prediction, which not only enhances the completeness of knowledge graphs but also serves as a measure of how effectively models can learn relational knowledge. We then provide a comprehensive overview and categorization of link prediction approaches and introduce the training and evaluation settings employed. Finally, we summarize common benchmark datasets used in the experimental studies presented in the subsequent chapters.

2.1 Knowledge Graphs

A *knowledge graph* (KG) is a multi-relational, directed graph describing facts about relationships between real-world entities. These facts are represented as (subject, predicate, object)-triples like (Nicholson, acted in, Departed). In such a graph, each node describes an entity, and each edge a relation. Next to the mere structure following a strict schema, KGs often come with rich textual features and descriptions (e.g., entity names, descriptions), numerical attributes (e.g., birthday), and even images (e.g., profile pictures). A visualization of an example KG can be seen in

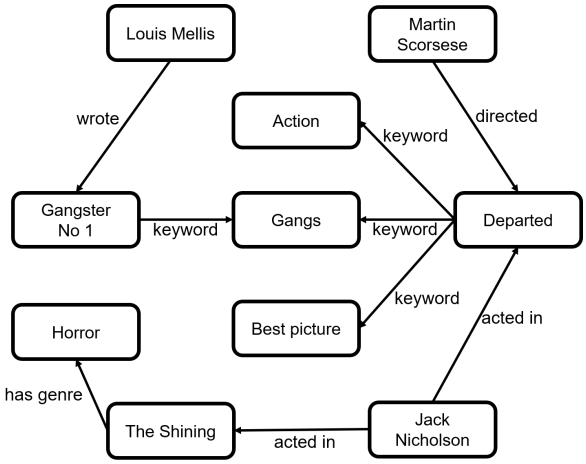


Figure 2.1: An example knowledge graph describing facts about movies.

Fig. 2.1. This modeling approach is used to provide structured knowledge of broad (e.g., Wikidata (Vrandečić and Krötzsch 2014), Yago (Suchanek et al. 2007)) as well as domain-specific sources (e.g., biomedical knowledge (Gene Ontology Consortium 2004; Robinson et al. 2008), machine parks (Yahya et al. 2021), and traffic rules (Monka et al. 2022)). The structured information can, for example, be helpful for predictive maintenance (Cao et al. 2022), autonomous driving (Monka et al. 2021), and enforcing factual correctness in large language models (Févry et al. 2020; Logan et al. 2019). For an in-depth overview of KGs, their modeling conventions, querying as well as reasoning approaches, see the survey presented by Hogan et al. (2021).

KGs are manually or automatically crafted following strict sets of rules. However, naturally many real-world systems can be described by a large number of interacting multi-typed components (Han 2009), e.g., human interactions, social networks, computer systems, and biological networks. When modeling these interactions as a graph, the resulting structure—termed *heterogeneous information network* (HIN) (Shi et al. 2016; Sun and Han 2013)—is similar to KGs. HINs and KGs mainly differ in the underlying data and slightly in the underlying schema. KGs describe a curated knowledge base of structured facts, and HINs mainly model interactions. Further, in HINs the nodes are typed, while in KGs types are represented as facts in the form of triples. This work focuses on KGs and shows the effectiveness of presented methods on KG-based datasets. However, due to the similarity of HINs and KGs, the presented methods can and are applied to HINs as well (El-Kishky et al. 2022). In the following, we do not distinguish between the two terms, but use the term KG to describe both data structures.

Notations	
a	scalar
$\boldsymbol{a}$	vector
$\boldsymbol{A}$	matrix
$\boldsymbol{\mathcal{A}}$	three-way tensor
$\mathcal{A}$	set
$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex numbers
Operations	
$\lvert\mathcal{A}\rvert$	size of set
$\lvert\lvert\boldsymbol{a}\rvert\rvert$	L2 norm
$\lvert\lvert\boldsymbol{a}\rvert\rvert_1$	L1 norm
$\bar{\boldsymbol{a}}$	elementwise complex conjugate

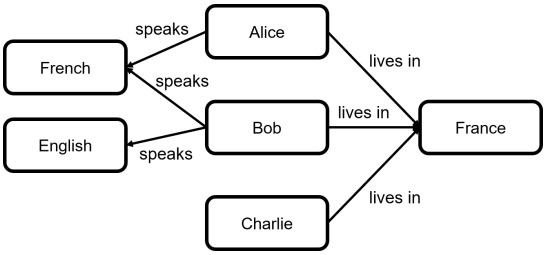
Table 2.1: Notations.

While KGs offer a structured set of facts, the knowledge stored in this structure is limited, and most graphs remain incomplete. The underlying unstructured data is vast, and the construction of a high-quality graph can be a large effort. Therefore, assumptions need to be made about knowledge not contained in the graph. Under the *closed* world assumption, all facts not described in the KG are treated as false. In contrast, following the *open* world assumption, such facts can either be true or false (Nickel et al. 2015). A middle ground between both is the *partial-closed* world assumption. Following this assumption, parts of the underlying data can be treated under the closed world assumption (complete), while others are assumed to be open (possibly incomplete). Following the open or partial-closed world assumption, we can reason about the yet undefined knowledge in the graph.

Throughout this thesis, we model a KG following the notations summarized in Tab. 2.1. A KG $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{K})$, with $\mathcal{K} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, is a collection of $N = |\mathcal{K}|$ subject-predicate-object (SPO) triples, where $\mathcal{E}$ denotes the set of entities and $\mathcal{R}$ the set of relations.

2.2 Link Prediction

Link prediction (LP) (Nickel et al. 2015) is the task of predicting missing facts in an incomplete KG in an automated fashion. Next to improving the completeness of a KG, the task of link prediction applies to recommendation systems (i.e., suggesting products/movies to users with similar tastes), social networks (e.g., friend suggestions), as well as protein-protein interaction networks (e.g., identifying new



(a)

(b)
Transductive (*CHARLIE, SPEAKS*, FRENCH)
Semi-inductive (***DAVE***, *SPEAKS*, FRENCH)
Inductive (***DAVE***, *SPEAKS*, **GERMAN**)

Figure 2.2: Example of a transductive, semi-inductive, and inductive query (italic) and expected answer (underlined) on a given graph. Entities not contained in the graph are marked in bold. In the semi-inductive setting, the new entity might come with additional information, such as connections to existing entities and/or text descriptions. In the inductive setting, a separate graph with disjunct entities but common relations might be provided during inference.

interaction based on known biological data). Alongside the direct application, it can serve as a proxy evaluating to what extent a model can process, store, and utilize structured information, which may be necessary for downstream applications; e.g., for drug discovery in a biomedical KG (Mohamed et al. 2019), for question answering (Huang et al. 2019; Saxena et al. 2020, 2022), query approximation (Hamilton et al. 2018), visual relationship detection (Baier et al. 2017), or replacing hand-crafted features in recommendation systems (El-Kishky et al. 2022).

LP is further subcategorized into three tasks; transductive (TD), semi-inductive (SI), and inductive link prediction. For a visualization of the three query types, see Fig. 2.2. *Transductive link prediction* is the task of predicting missing facts between existing entities in the KG. *Semi-inductive link prediction* focuses on modeling entities that are not yet part of the graph, and connecting them to existing entities; e.g., a new user joining a social network, a new product, or event. Here, the new entity typically comes with additional information such as connections to existing entities and/or text descriptions. To address this task without retraining the complete model (which can be expensive), models need to be able to process new additional input information during inference. *Inductive link prediction* reasons about a new KG with completely separate entities (but shared relations). To address this task, models need to reason about the new input graph during inference. Ch. 3-5 focus on transductive and Ch. 6 on semi-inductive link prediction.

	Model Type	Text	Models
Local	Bi-Encoder		RESCAL (Nickel et al. 2011), TransE (Bordes et al. 2013), DistMult (Yang et al. 2015), ComplEx (Trouillon et al. 2016), SimplE (Kazemi and Poole 2018), RotatE (Sun et al. 2019), TuckER (Balažević et al. 2019), MEIM (Tran and Takasu 2022)
		✓	*DKRL* (Xie et al. 2016), *KEPLER* (Wang et al. 2021b), *BLP* (Daza et al. 2021), *StaR* (Wang et al. 2021c), *SimKGC* (Wang et al. 2022)
	Cross-Encoder	✓	*KG-Bert* (Yao et al. 2019)
	Encoder-Decoder	✓	*KGT5* (Ch. 5)
Global	GNN		RGCN (Schlichtkrull et al. 2018), *NBFNet* (Zhu et al. 2021)
	Rule-based		Amie (Galárraga et al. 2013), AnyBURL (Meilicke et al. 2019), LERP (Han et al. 2023)
Hybrid	Bi-Encoder		HittER (Chen et al. 2021)
		✓	*SimKGC* (Wang et al. 2022)
	Encoder-decoder	✓	*KGT5-context* (Ch. 5)

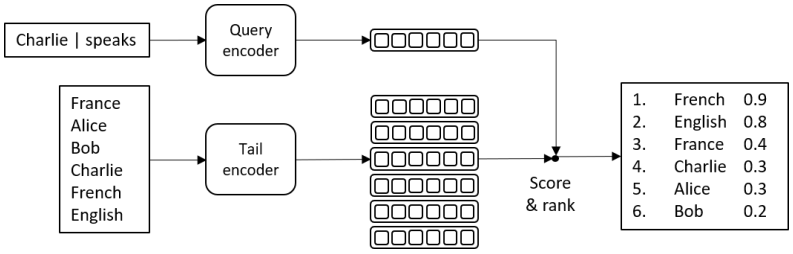
Table 2.2: Categorization of link prediction approaches. Parametric approaches (marked in italics) scale independently of the graph size.

2.3 Categorization of Link Prediction Approaches

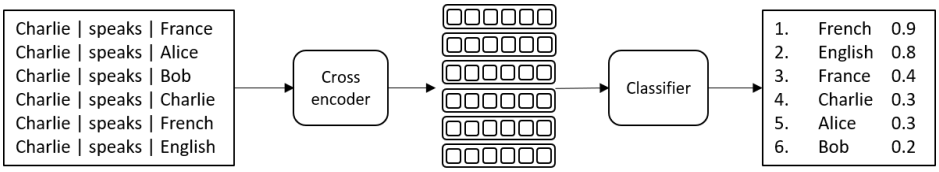
Tab. 2.2 gives an overview of common link prediction approaches. The models can mainly be grouped into two categories, *global* and *local* models. Global models operate on the full graph for inference, such as GNN-based and rule-based approaches. Local models operate on a triple level. They can be further subcategorized into *bi-encoder*, *cross-encoder*, and *encoder-decoder* models. For a visualization of the three subcategories, see Fig. 2.3. However, some models fall in between the two main categories, global and local. We call those models *hybrid*. Note that for all visualizations in this section, we use the example KG presented in Fig. 2.2 together with the running example query (`Charlie, speaks, ?`).

2.3.1 Local Models

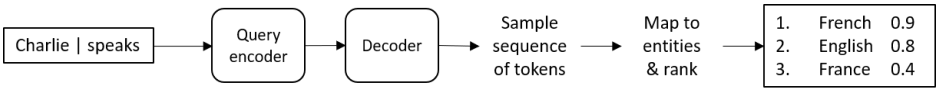
Bi-encoder models (Fig. 2.3a) consist of two separate encoder modules, a query encoder f_{qe}, and a target encoder f_{te}. Given representations $\boldsymbol{s}$ and $\boldsymbol{p}$ of the query entity s and relation p (and optional additional query-specific features), bi-encoder models create a combined representation of the query via the query encoder—$f_{qe}(\boldsymbol{s}, \boldsymbol{p})$. Typically, the target encoder is applied to all possible answers, i.e., $\mathcal{T} = \{f_{te}(\boldsymbol{e}) \mid$



(a) Bi-encoder.



(b) Cross-encoder.



(c) Encoder-decoder.

Figure 2.3: Inference of local models visualized for the transductive example query of Fig. 2.2. (a) A bi-encoder encodes query and target separately. For inference, all possible targets are encoded and scored against the query via the model-specific scoring function. (b) A cross-encoder encodes the query together with each possible target. Inference can be performed on a triple basis, i.e., by classifying whether the triple exists, or by calculating a probability distribution over all possible encoded query-target pairs. (c) An encoder-decoder assumes each entity to be represented by a sequence of tokens. It encodes the query, and a decoder generates sequences that are mapped to candidate entities.

$e \in \mathcal{E}\}$. To gauge the plausibility of a possible answer, bi-encoder models define a scoring function that combines the query ($\boldsymbol{qe}$) and target representation ($\boldsymbol{t}$). For a ranking of all possible answers, this scoring function is applied to all resulting target representations $f_{score}(\boldsymbol{qe}, \boldsymbol{t}) \forall \boldsymbol{t} \in \mathcal{T}$.

Cross-encoder models (Fig. 2.3b) offer a holistic view and encode the query and target in a combined fashion. They define the scoring function to gauge the plausibility of a possible answer candidate c as $f_{score}(\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{c})$. This combined encoding allows for more fine-grained interaction between the query features and the answer candidate, but leads to increased inference cost, as the query needs to be encoded together with each possible answer candidate.

Encoder-decoder models (Fig. 2.3c) define a query encoder $f_{qe}(\boldsymbol{s}, \boldsymbol{p})$ similar to the bi-encoder approaches. However, they replace the target encoder with a generative decoder. Such an architecture is mainly applicable if an entity can be represented by a sequence of representations; e.g., textual input such as entity mentions. Consider

an entity with the mention "black hole". For this example, the mention could be represented by the sequence of tokens "black" and "hole". The model defines a vocabulary consisting of a fixed number of such tokens. For generation, we sample step by step from the distribution over the tokens conditioned on the input query.

Knowledge Graph Embeddings

Knowledge graph embedding (KGE) models are non-parametric bi-encoder models (Fig. 2.3a). They associate an *embedding* with each entity and each relation; the embeddings are taken from a vector space specific to the respective KGE model. Additionally, they define model-specific query- and target-encoders, as well as scoring functions. For most KGE models, however, the target encoder is defined by an embedding layer only. Most KGE models are trained and applied in a transductive link prediction setting. For a semi-inductive setting, extensions are needed; see Ch. 6.

KGE models can mainly be grouped into two families; *translational distance* and *semantic matching* models. In the following, we describe the subset of KGE models utilized throughout this study.

Translational Distance Models

Translational distance models define the query encoding as a relation-specific translation of the query entity and measure the distance of the translated query entity to the target.

TransE (Bordes et al. 2013) is a simple and one of the most popular translational distance models. Both, entities and relations are embedded into the same space. It defines the translation as a sum and the distance function as the Euclidian distance. Given a triple (s, p, o), and its corresponding representations $\boldsymbol{s}$, $\boldsymbol{p}$, and $\boldsymbol{o}$, its query encoder and scoring function are

$$f_{qe}(\boldsymbol{s}, \boldsymbol{p}) = \boldsymbol{s} + \boldsymbol{p}, \qquad f_{score}(\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o}) = -||f_{qe}(\boldsymbol{s}, \boldsymbol{p}) - \boldsymbol{o}|| \tag{2.1}$$

A triple has high plausibility if the translated subject representation is close to the object representation. However, the proposed translation approach implemented by the sum operation cannot represent *symmetric* relations, such as `marriedTo`.

RotatE. Addressing the limitations of TransE representing symmetric relations, RotatE (Sun et al. 2019) defines the translation operation as a rotation from the subject to the object entity in complex space. This translation is implemented via the Hadamard product ($\circ$) between complex valued representations. Its query encoder and scoring function are

$$f_{qe}(\boldsymbol{s}, \boldsymbol{p}) = \boldsymbol{s} \circ \boldsymbol{p}, \qquad f_{score}(\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o}) = -||f_{qe}(\boldsymbol{s}, \boldsymbol{p}) - \boldsymbol{o}||_1 \tag{2.2}$$

where $\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o} \in \mathbb{C}^d$.

Further examples of translational distance models are BoxE (Abboud et al. 2020) and PairRE (Chao et al. 2021).

Semantic Matching Models

Semantic matching models use similarity functions to gauge the existence of a triple. A subfamily of semantic matching models is bilinear models. These models treat the representation learning process as a tensor factorization. Considering the KG as a three-way adjacency tensor indicating connections between entities via relational slices, bilinear models learn to represent this sparse tensor via dense representations. This factorization approach can be described as

$$\boldsymbol{\mathcal{K}} = \boldsymbol{E}\boldsymbol{\mathcal{R}}\boldsymbol{E}^T, \tag{2.3}$$

where $\boldsymbol{\mathcal{K}} \in \mathbb{N}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$, $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{E}|\times d}$, $\boldsymbol{\mathcal{R}} \in \mathbb{R}^{|\mathcal{R}|\times d\times d}$, and $d \ll |\mathcal{E}|$

DistMult (Yang et al. 2015), also called bilinear diagonal model, performs tensor factorization with constraints. It restricts the matrix representing a relation i to a diagonal matrix, i.e., $\boldsymbol{\mathcal{R}}_{[i]} = diag(\boldsymbol{p})$, where $\boldsymbol{p} \in \mathbb{R}^d$ and $\boldsymbol{\mathcal{R}}_{[i]} \in \mathbb{R}^{d\times d}$. Its scoring function to gauge the plausibility of a single triple (an entry in $\boldsymbol{\mathcal{K}}$) is

$$f_{qe}(\boldsymbol{s}, \boldsymbol{p}) = \boldsymbol{s}^T diag(\boldsymbol{p}), \qquad f_{score}(\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o}) = f_{qe}(\boldsymbol{s}, \boldsymbol{p})\boldsymbol{o} \tag{2.4}$$

While simple, this approach comes with limitations in representing relational patterns. Due to the symmetric property of the dot product used in the scoring function, the model is not able to represent *antisymmetric* relationship types; e.g., `childOf`.

ComplEx. In contrast to DistMult, ComplEx (Trouillon et al. 2016) increases the number of representable relation patterns by utilizing complex valued embeddings. It uses the score function

$$f_{qe}(\boldsymbol{s}, \boldsymbol{p}) = \boldsymbol{s}^T diag(\boldsymbol{p}), \qquad f_{score}(\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o}) = Re(f_{qe}(\boldsymbol{s}, \boldsymbol{p})\bar{\boldsymbol{o}}) \tag{2.5}$$

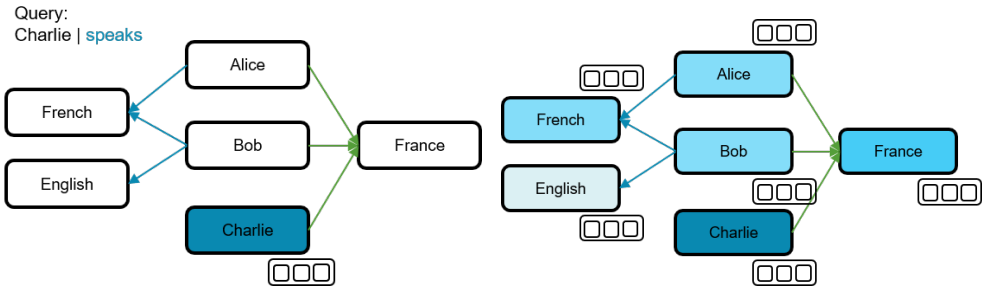
where $\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{o} \in \mathbb{C}^d$ and $\mathrm{Re}(x)$ refers to the real part of $x \in \mathbb{C}$ and $\bar{\boldsymbol{x}}$ to the element-wise complex conjugate of $\boldsymbol{x} \in \mathbb{C}^d$.

Further examples of semantic matching models are RESCAL (Nickel et al. 2011), SimplE (Kazemi and Poole 2018), QuatE (Zhang et al. 2019), TuckER (Balažević et al. 2019), and MEIM (Tran and Takasu 2022).

Text-Based Link Prediction

KGE models utilize the underlying graph structure only without considering any additional information. However, multiple graphs are enriched with text information such as entity and relation mentions and even descriptions. Numerous approaches try to integrate both information sources, structural and textual information. One straightforward integration of text into existing architectures is to initialize KGE models with word embeddings; an approach taken by DKRL (Xie et al. 2016). In this setting, the textual encoding is performed separately for subject, relation, and object. To allow for interaction of text information stemming from the query entity and relation, the bi-encoder SimKGC (Wang et al. 2022) defines the query encoder as a BERT Transformer (Devlin et al. 2019) encoding the combined sequence of query entity and relation (e.g., "Charlie | speaks", c.f. Fig. 2.3a). In addition, it defines a separate Transformer as the target encoder to capture the textual information for the answer candidates. Offering a holistic view of the query and possible answer, the cross-encoder KG-BERT (Yao et al. 2019) encodes the textual information of the query together with the target (e.g., "Charlie | speaks | French", c.f. Fig. 2.3b). However, this holistic view comes with high inference cost, as the query needs to be encoded together with each possible target. Finally, the encoder-decoder approach KGT5 (Ch. 5) treats link prediction as a sequence-to-sequence task. As SimKGC, it utilizes a Transformer-based query-encoder but replaces the target encoder with a decoder. The decoder generates mentions corresponding to possible answer entities (c.f. Fig. 2.3c). For a more detailed overview of text-based link prediction approaches, see Ch. 5.

With these text-based approaches, entities are represented by the sequence of (sub-)words building the entity mention. In contrast to KGE models, they do not learn a representation per unique entity but rather per unique subword. Therefore, they are parametric, and the model size scales with the size of this vocabulary $\mathcal{V}$ over sub-words. Typically, for large graphs $|\mathcal{V}| \ll |\mathcal{E}|$ holds. Consequently, the model size is considerably smaller compared to KGE models. Further, as entities are represented based on their textual input, most text-based approaches can represent emerging entities and can therefore be applied in the transductive, semi-inductive, and in some cases even inductive setting.



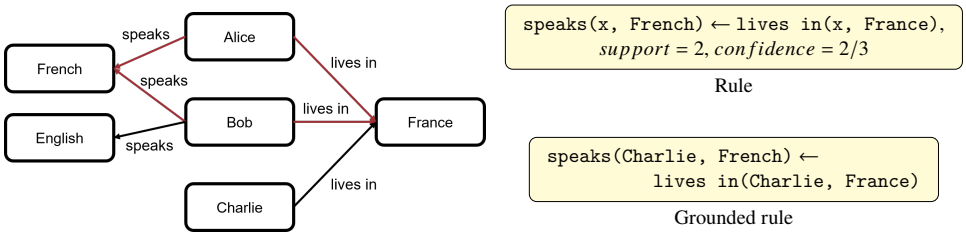
(a) NBFNet initialization. **(b)** NBFNet propagation.

Figure 2.4: (a) NBFNet initializes the query entity with the initial representation of the query relation (marked in blue). (b) This representation is propagated through the graph. Each entity results with a query-specific representation, which is used to score the possible answer candidates using the model-specific scoring function.

2.3.2 Global Models

GNN-Based Link Prediction

GNNs (Kipf and Welling 2016) are a prominent architecture choice for link prediction in graphs. They encode nodes by their multi-hop neighborhood and update node representations by aggregating information from a node's neighbors through learnable functions. However, they do not make use of information stemming from multiple different relation types. RGCNs (Schlichtkrull et al. 2018) incorporate multiple relationship types by using separate aggregation operations for different types of relationships in the graph. While this approach is promising for tasks like node classification, performance on link prediction tasks stays behind other approaches like KGEs. Focusing on link prediction, Zhu et al. (2021) introduced NBFNet. NBFNet does not learn entity embeddings but calculates query-specific representations based on layer-specific relation transformations. This procedure is visualized in Fig. 2.4 for the example query (`Charlie, speaks, ?`). The representation of the query entity (`Charlie`) is initialized with parameters specific to the query relation (`speaks`). This representation is then propagated through the graph, so that each entity results with a query-specific representation, capturing the relation-specific interconnections to the query entity through the graph. To gauge the plausibility of a triple, the resulting entity representations of query and target are scored similarly to semantic matching KGEs. This approach can perform on par or even outperform KGEs for link prediction (see Tab. 2.4). As NBFNet calculates entity representations per query on a provided graph, it offers flexibility for changing graphs. It allows for transductive, semi-inductive, and inductive link prediction. Additionally, in contrast to KGEs and RGCNs, NBFNet is parametric. It does not scale with the size of the graph, but by $|\mathcal{R}| \times d$. However, for



(a) All occurrences of a sampled path marked in red. **(b)** Resulting rule (top) and grounded rule for inference (bottom).

Figure 2.5: Rule mining for link prediction. Sample paths over a knowledge graph, count occurrences for support and calculate confidence to build a set of rules.

inference, all entity representations are materialized resulting in a memory footprint of $(|\mathcal{E}| + |\mathcal{R}|) \times d$.

Rule-Based Link Prediction.

Next to neural, rule-based approaches can be used for link prediction. Here, one of the currently best-performing approaches is AnyBURL (Meilicke et al. 2019). It mines rules by sampling paths and counting individual occurrences in the KG. For an exemplified visualization for the query (`Charlie, speaks, ?`), see Fig. 2.5. When no representations for further downstream tasks are needed, such rule-based approaches are a lightweight alternative to neural ones, and even establish the state-of-the-art on multiple benchmarks (see Tab. 2.4).

Here, the model size is dependent on the number of learned rules. In contrast to neural approaches, rule-based models are defined by a set of rules (Fig. 2.5b top). However, approaches such as AnyBURL (Meilicke et al. 2019) learn millions of rules even for small graphs to reach a high link prediction performance. In these cases, storage cost might even be higher than for neural approaches. But the number of rules is (rather) independent of graph size. Therefore, on large-scale graphs, storage overhead is typically considerably lower compared to most neural approaches.

2.3.3 Hybrid Models

Some models cannot be clearly categorized into global or local models. Those models are mainly extensions of local models that query the graph to encode a subgraph together with either query or target. Consider the input to the query encoder in Fig. 2.3: "`Charlie | speaks`". The input together with the subgraph consisting of the one-hop neighborhood of the query entity could, for example, be "`Charlie | speaks`

`context: lives in | France`".[a] Such an approach is taken by the Transformer-based bi-encoder HittER (Chen et al. 2021). It encodes the query together with embeddings/features of sampled neighbors of the one-hop neighborhood of the query entity. Similarly, the encoder-decoder approach KGT5-context (Ch. 5) extends KGT5 with HittER's neighborhood integration idea for raw-text feature input. Further, SimKGC (Wang et al. 2022) can be extended to encode the raw-text feature input of the query entity's, as well as target entity's one-hop neighborhood. Due to the high cost, Wang et al. (2022) only apply this extension on small graphs.

2.4 Training

One popular training objective for link prediction models is separating *positive* (i.e., correct) triples from *negative* (incorrect) triples. Here, models aim to assign high scores to positive triples and low scores to negative ones. They are trained using gradient-based optimization. The cost function is composed of the mean loss (e.g., binary cross entropy) taken over each positive triple $t \in \mathcal{K}$ and its associated negative examples $\mathcal{S}^-(t)$—i.e, $|\mathcal{K}|^{-1} \sum_{t \in \mathcal{K}} \text{loss}(t, \mathcal{S}^-(t))$—and regularization terms. However, the knowledge graph provides only positive but no negative triples. Therefore, training often relies on *negative sampling* to provide *pseudo-negative* examples, i.e., triples that are likely but not guaranteed to be actual negatives.

2.4.1 Negative sampling

A common and effective method to obtain informative pseudo-negative triples is to corrupt the subject or object (and sometimes also the relation) of known positive triples. For example, the positive triple (Hitchcock, directed, Psycho) can be corrupted by replacing the object with some other entity obtaining, say, the pseudo-negative triple (Hitchcock, directed, Avatar). Pseudo-negatives are commonly generated using *negative sampling*, in which the replacement entity is sampled randomly from the set of entities. In the simplest setting, the replacement entity is sampled uniformly from the set of all entities $\mathcal{E}$. However, this set can be predefined, e.g., by restricting it to the entities occurring in a training batch. Additionally, the distribution can be adapted; e.g., sampling by the frequency of entities. For a more detailed overview of sampling methods and their effects on large-scale training, see Sec. 3.4. Each positive triple is used to generate multiple pseudo-negative samples; the number of such samples is a hyperparameter. Note that an alternative approach

[a] Here presented with raw text input. The input can also be the embeddings/features corresponding to the mentioned entities and relations.

Algorithm 1 Basic bi-encoder training.

Require:
$\mathcal{G}$ - KG defined by a set of entities $\mathcal{E}$, relations $\mathcal{R}$, and triples $\mathcal{K}$,
N_s^- - num. negatives (subject),
N_o^- - num. negatives (object),
Ensure: M - Model

1: **for each** epoch **do**
2: **for each** $\mathcal{B} \in \texttt{batch}(\mathcal{K})$ **do**
3: $\mathcal{S}^- \leftarrow \emptyset$
4: **for each** $(s, p, o) \in \mathcal{B}$ **do**
5: $\mathcal{E}_s^-, \mathcal{E}_o^- \leftarrow \texttt{sample\_negatives}(N_s^-, N_o^-, \mathcal{E})$
6: $\mathcal{S}^- \leftarrow \mathcal{S}^- \cup \{(s', p, o)$ **for each** $s' \in \mathcal{E}_s^-\}$ ▷ Corrupt subject
7: $\mathcal{S}^- \leftarrow \mathcal{S}^- \cup \{(s, p, o')$ **for each** $o' \in \mathcal{E}_o^-\}$ ▷ Corrupt object
8: **end for**
9: $scores \leftarrow M.\texttt{score}(\mathcal{B} \cup \mathcal{S}^-)$ ▷ Score positives and negatives
10: $loss \leftarrow \texttt{compute\_loss}(scores, \mathcal{B} \cup \mathcal{S}^-)$
11: $M \leftarrow \texttt{update\_parameters}(M, loss)$
12: **end for**
13: **end for**

to sampling is to use all entities from $\mathcal{E}$ for corruption. Although this "1vsAll" method (Dettmers et al. 2018) can be very effective on smaller KGs, we do not consider it further because it does not scale to large KGs with millions of entities.

2.4.2 Training Procedure

Bi- & cross-encoder training. The general training approach for bi- and cross-encoder models is summarized in Alg. 1. Processing is performed in mini-batches of batch size B (e.g., 1024, line 2). For each triple in the batch, $N^- = N_s^- + N_o^-$ pseudo-negative examples are generated by corrupting the subject $N_s^-\times$ and the object $N_o^-\times$ (line 4-8). Each batch thus consists of B positive and BN^- pseudo-negative triples. Batches are processed in parallel (on a GPU). This involves computing the batch loss (forward pass, line 10), its gradient (backward pass) and updating model parameters (line 11) using an optimizer such as Adagrad (Duchi et al. 2011) or Adam (Kingma and Ba 2015). Note that GNN-based training for models such as NBFNet is similar to the described procedure, but before scoring the triples (line 9), query-specific entity representations need to be calculated.

Encoder-decoder training. A common approach for training generative models like encoder-decoder is teacher forcing (Williams and Zipser 1989). Here, the training objective is to generate the next correct token corresponding to the sequence representing the answer entity of a query. Therefore, instead of aiming for a high score for

correct triples and a low one for incorrect triples, as for bi-encoders, we aim for a high probability for the correct token and a low probability for incorrect tokens. This can be seen as a "1vsAll" training approach, scoring against all tokens in the vocabulary in each generation step, in contrast to scoring against all entities. For more details on encoder-decoder training, see Ch. 5.

Training cost. During each epoch of training a bi-encoder model, all positives and their associated negatives are scored, i.e., the overall number of per-epoch score computations is $(|\mathcal{K}| + 1)N^-$. We use this number as a proxy for computational cost throughout. As the architecture of most KGE models is slim, with the encoder being an embedding layer only, scoring a single triple is cheaper compared to text-based approaches. Therefore, the text-based approaches are typically trained for considerably fewer epochs and using fewer negative examples (e.g., see (Wang et al. 2022)). As encoder-decoder approaches are trained via teacher forcing, training is independent of the negatives. Here, cost scales by the vocabulary size and sequence length of the entity representations. Rule-based methods do not learn representations, but sample rules over the graph. For large graphs, training is typically cheaper compared to neural approaches.

2.4.3 Hyperparameters

The resulting model quality, especially for KGE models, is highly dependent on selected hyperparameters (Ali et al. 2021a; Ruffinelli et al. 2020). In addition, the choice of hyperparameters is model- and dataset-dependent. Therefore, training comes with high cost for hyperparameter tuning. Important hyperparameters include embedding dimensionality, training type (e.g., negative sampling, 1vsAll), number N^- of negatives, sampling type (e.g., uniform, frequency-based), loss function (e.g., cross entropy, margin loss), optimizer (e.g., Adam, Adagrad), learning rate, type and weight of regularization (e.g., weighted, unweighted, L2, L3), and amount of dropout.

The hyperparameter space for KGE models is discussed in detail in Ali et al. (2021a) and Ruffinelli et al. (2020).

2.4.4 Frameworks

Frameworks such as OpenKE (Han et al. 2018), Ampligraph (Costabello et al. 2019), PyKeen (Ali et al. 2021b) or LibKGE (Broscheit et al. 2020) provide implementations of various KGE models as well as training algorithms, evaluation methods, and hyperparameter tuning approaches. Further, Pytorch BigGraph (Lerer et al. 2019), DGL-KE (Zheng et al. 2020), and Dist-KGE (Ch. 3) provide KGE implementations

Algorithm 2 Evaluation - entity ranking.

Require:
$\mathcal{G}$ - KG defined by set of entities $\mathcal{E}$, relations $\mathcal{R}$, and triples $\mathcal{K}$,
$\mathcal{K}^{eval}$ - Hold-out evaluation set,
M - trained model
Ensure: m - Evaluation metric aggregated over $\forall t \in \mathcal{K}_{test}$

1: $\mathcal{Q} \leftarrow \emptyset$ ▷ Set of ranks
2: **for each** $(s, p, o) \in \mathcal{K}^{eval}$ **do**
▷ tail prediction
3: $\mathcal{C}_o \leftarrow \{(s, p, o') \textbf{ for each } o' \in \mathcal{E}\}$ ▷ Tail candidate triples
4: $scores_o \leftarrow M.\texttt{score}(\mathcal{C}_o)$
5: $scores_o \leftarrow \texttt{filter\_out\_true\_answers}(\mathcal{K}, scores_o)$
6: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \texttt{get\_rank}(o, scores_o, \mathcal{E})$
▷ head prediction
7: $\mathcal{C}_s \leftarrow \{(s', p, o) \textbf{ for each } s' \in \mathcal{E}\}$ ▷ Head candidates triples
8: $scores_s \leftarrow M.\texttt{score}(\mathcal{C}_s)$
9: $scores_s \leftarrow \texttt{filter\_out\_true\_answers}(\mathcal{K}, scores_s)$
10: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \texttt{get\_rank}(s, scores_s, \mathcal{E})$
11: **end for**
12: $m \leftarrow \texttt{compute\_metric}(\mathcal{Q})$

and training approaches for large-scale graphs. Rule-based approaches are implemented in PyClause (Betz et al. 2024).

2.5 Evaluation

The standard approach to evaluate model quality for the link prediction task is to use the *entity ranking* protocol (Ji et al. 2021; Wang et al. 2017) summarized in Alg. 2. This protocol asks models to answer the queries $(s, p, ?)$ (line 3-6) and $(?, p, o)$ (line 7-10) for all triples of a held-out evaluation set. As multiple entities could be a correct answer, models need to generate multiple ranked candidates. Usually, all entities in $\mathcal{E}$ are scored to compute a ranking. Given the ranked answer candidates, common metrics reported over the ranked candidates are the mean reciprocal rank (MRR) and Hits@k. For the MRR, we extract the reciprocal rank of the true answer and take the mean over all queries (Eq. 2.6). Hits@k describes the number of times the true answer has rank $\leq k$ normalized by the number of queries $|2\mathcal{K}^{test}|$ (Eq. 2.7). To avoid punishing a model for ranking a true but already known answer high, we filter out all candidate triples that occur in the training, validation or test data before determining the reciprocal rank or Hits@k of the test triple itself (line 5 & 9). In

the case of ties, we use the mean rank to avoid misleading results (Rossi et al. 2021; Ruffinelli et al. 2020; Sun et al. 2020).

$$\text{MRR} = \frac{1}{2|\mathcal{K}^{eval}|} \sum_{(s,p,o)\in\mathcal{K}^{\text{eval}}} \left(\frac{1}{rank(s|p,o)} + \frac{1}{rank(o|s,p)} \right), \tag{2.6}$$

$$\text{Hits@}k = \frac{1}{2|\mathcal{K}^{eval}|} \sum_{(s,p,o)\in\mathcal{K}^{\text{eval}}} (\mathbb{1}(rank(s|p,o) \leq k) + \mathbb{1}(rank(o|s,p) \leq k)) \tag{2.7}$$

Evaluation cost. For each (s, p, o)-triple in a held-out test set $\mathcal{K}^{\text{test}}$, this protocol requires to score all triples of form $(s, p, ?)$ and $(?, p, o)$ using all entities in $\mathcal{E}$. Overall, $2|\mathcal{K}^{\text{test}}||\mathcal{E}|$ scores are computed so that evaluation cost scales linearly with the number of entities. Since this cost can be substantial, sampling-based approximations, termed sMRR, have been used in some prior studies (Lerer et al. 2019; Zheng et al. 2020). In this case, the MRR is not reported over all ranked candidates but only a sampled subset of candidates. In Ch. 3, we show that such approximations can be misleading in that they do not reflect model quality faithfully. However, in general, this inference setting is related to maximum inner product search (MIPS) (Abuzaid et al. 2019; Aguerrebere et al. 2023; Douze et al. 2024; Johnson et al. 2019), and corresponding algorithms can be directly applied to many bi-encoder models to find top scoring candidates with $\ll |\mathcal{E}|$ score calculations. However, the global NBFNet model calculates query-specific entity representations. It is infeasible to precompute these representations and, therefore, MIPS is not advantageous. Inference cost of a generative encoder-decoder approach is independent of the number of entities in the graph. Here, inference cost depends on the vocabulary size, depth of the decoder, and average number of tokens used per entity representation. For rule-based approaches, inference cost depends on the number of rules, number of entities, and length of the rule. Applying all relevant rules (see Fig. 2.5b bottom) and aggregating their confidences to rank all possible candidates can be more expensive than neural approaches.

2.6 Datasets

Throughout this work, we make use of a wide range of benchmark datasets introduced over the last decade. Statistics of presented datasets are summarized in Tab. 2.3 and the current state-of-the-art per dataset in Tab. 2.4.

FB15K introduced by Bordes et al. (2013) is a subset of the knowledge graph Freebase (Bollacker et al. 2008). It contains almost 15 000 unique entities and over 1 000

Dataset	Entities	Relations	|Train|	|Valid|	|Test|
FB15K	14 951	1 345	483 142	50 000	59 071
FB15K-237	14 505	237	272 115	17 535	20 466
WN18	40 943	18	141 442	5 000	5 000
WNRR	40 559	11	86 835	3 034	3 134
Yago3-10	123 182	37	1 079 040	5 000	5 000
Wikidata5M	4 818 579	828	21 343 681	5 357	5 321
Freebase	86 054 151	14 824	304 727 650	16 929 318	16 929 308
WikiKG90Mv2	91 230 610	1 387	601 062 811	15 000	10 000

Table 2.3: Dataset statistics.

Dataset	Text	Model	Paper	MRR	Hits @10
FB15K		ComplEx	(Lacroix et al. 2018)	0.860	0.910
WN18		Inverse Model	(Dettmers et al. 2018)	0.963	0.964
FB15K-237		NBFNet	(Zhu et al. 2021)	0.415	0.599
	✓	SimKGC	(Jiang et al. 2023)	0.367	0.543
WNRR		LERP	(Han et al. 2023)	0.622	0.682
	✓	SimKGC	(Jiang et al. 2023)	0.692	0.811
Yago3-10		MEIM	(Tran and Takasu 2022)	0.585	0.716
	✓	KGT5+ComplEx	(Saxena et al. 2022)	0.552	0.680
Wikidata5M		AnyBURL	(Meilicke et al. 2024)	0.353	0.433
	✓	KGT5-context	(Kochsiek et al. 2023)	0.426	0.460
Freebase		AnyBURL	(Meilicke et al. 2024)	0.711	0.728
WikiKG90Mv2	✓	KGT5-context	(Kochsiek et al. 2023)	0.301[b]	-

Table 2.4: Current state-of-the-art per dataset. For a categorization of each listed model see Tab. 2.2.

relations with a large fraction of the content being about movies, actors, awards, sports and sport teams. Bordes et al. (2013) selected only entities occurring in the Wikilinks database that have ≥ 100 mentions in Freebase.

FB15K-237 introduced by Toutanova and Chen (2015) is an adaption of the dataset FB15K. The authors realized that a large part of test queries could be answered by a near identical relation or near inverse relation observed in the training data. E.g., the test set contains the triple

[b] Valid-MRR instead of test-MRR. KGT5-context directly used mentions of entities and relations for WikiKG90Mv2, instead of the textual embeddings used by other models. For this reason, the benchmark authors (Hu et al. 2021) did not provide scores on the hidden test set. Mentions are provided with the dataset.

(HELLO DOLLY!, AWARD_NOMINATION/AWARD, AWARD FOR BEST ORIGINAL MUSICAL),

while the train set contains the triple

(AWARD FOR BEST ORIGINAL MUSICAL, AWARD_NOMINATION/NOMINATED_FOR, HELLO DOLLY!).

To make the task more challenging, Toutanova and Chen (2015) removed infrequent relations, near duplicate, and near inverse relations. This change reduced the number of relations from $\approx$ 1 000 to 237. Additionally, from the validation and test sets, they filtered any triples whose entity pairs were directly linked in the training set. While such cases could be present in a realistic scenario, they were excluded to avoid additional trivial cases.

WN18 introduced by Bordes et al. (2014) is a subset of the KG Wordnet, designed to produce an intuitively usable dictionary and thesaurus. This dataset uses 18 relations of the full graph and selects all connected entities occurring at least 15×. This results in over 40 000 entities. An example triple of this KG is (STATISTICALREGRESSION, PARTOF, REGRESSIONANALYSIS).

WNRR introduced by Dettmers et al. (2018) is an adaption of the dataset WN18. Similar to FB15K, WN18 contains a large number of near inverse relations; e.g., the relations PARTOF and HASPART. Specifically, learning a rule to identify inverse relations and using this rule for inference by checking the query's inverse relation in the training set has led to the most effective method on WN18 so far. This rule achieved the current highest MRR on WN18, see Tab. 2.4. The authors followed the approach taken for the creation of FB15K-237 to create the more challenging subset WNRR. This procedure reduced the number of relations from 18 to 11.

Yago3-10 introduced by Dettmers et al. (2018) is a subset of the KG YAGO3 (Mahdisoltani et al. 2015). This subset is constructed so that each entity has a degree of $\geq$ 10. It consists of over 120 000 entities, 37 relations, and over 1 000 000 triples. Most triples deal with descriptive attributes of people, such as (ALBERTEINSTEIN, WASBORNIN, ULM)

Wikidata5M introduced by Wang et al. (2021b) is a subset of the KG Wikidata (Vrandečić and Krötzsch 2014). A subset of $\approx 5M$ entities is extracted by discarding all entities, which either cannot be mapped to a Wikipedia page or have a Wikipedia description consisting of < 5 words. Similarly, only relations with non-empty Wikipedia pages are kept resulting in 822 relations. Next to the pure graph structure, Wikidata5M additionally offers text-based mentions, as well as Wikipe-

dia descriptions alongside each entity. This allows for evaluation of text-based link prediction and KGE models.

Freebase. The full KG constructed by Bollacker et al. (2008) is used and provided by Zheng et al. (2020) as a large-scale benchmark for link prediction. With more than $80M$ entities and $300M$ triples, this benchmark aims to simulate large-scale industry settings.

WikiKG90Mv2 introduced by Hu et al. (2021) is a large-scale benchmark with a public leaderboard. It is based on a dump of Wikidata of the year 2021. Next to the graph structure, this benchmark comes with pretrained MPNet (Song et al. 2020) embeddings based on entity mentions and descriptions. While raw text entities and descriptions are provided, the use of this data is not allowed for competition on the public leaderboard.

Chapter 3

Training of Large-Scale Knowledge Graph Embedding Models

"Scaling laws are decided by god; the constants are determined by members of the technical staff."

Sam Altman, 2024

Knowledge graphs scale up to millions (e.g., Freebase) or even billions (e.g., social networks) of nodes. However, a large part of the available literature focuses on small KGs. Recently, several frameworks that are able to train KGE models for large-scale KGs by parallelization across multiple GPUs or machines have recently been proposed. So far, the benefits and drawbacks of the various parallelization techniques have not been studied comprehensively. In this chapter, we report on an experimental study in which we presented, re-implemented in a common computational framework, investigated, and improved the available techniques.

This chapter is based on Kochsiek and Gemulla (2021).

3.1 Introduction

Large-scale knowledge graphs contain millions of entities, which makes training KGE models challenging. The Freebase (Bollacker et al. 2008) KG, for example, contains more than 80M entities and 100s of millions of triples, leading to large model sizes and long training times. Even relatively low-dimensional KGE models may exceed the

memory of GPUs (e.g., a ComplEx model with $d = 400$ takes 128 GB plus optimizer state). A more extreme example is Bess (Cattaneo et al. 2022). The authors train an ensemble of 85 KGE models with a total parameter count of ≈ 2.6 T, highlighting the necessity of efficient and effective parallel training and parameter handling.

However, a large part of the available literature focuses on small KGs, and training of such large-scale KGE models is only possible due to recently proposed parallelization approaches; namely the frameworks PyTorch BigGraph (Lerer et al. 2019), GraphVite (Zhu et al. 2019), and DGL-KE (Zheng et al. 2020). These frameworks have enabled large-scale industry applications of KGEs. For instance, El-Kishky et al. (2022) utilized KGEs trained with PyTorch BigGraph on a large-scale social network for downstream recommendation tasks, effectively replacing numerous handcrafted features with the trained embeddings.

Each of these frameworks introduced distinct parallelization techniques. This chapter presents an independent investigation into the efficiency and effectiveness of these techniques within a unified framework.

The primary challenge in parallelizing KGE training lies in the synchronization of entity and relation embeddings across workers. Accesses to these embeddings are required both when processing a triple from the KG but also in the negative sampling step (see Fig. 3.2). Each access potentially involves communication between workers, which is costly. To reduce this cost, parallel training methods (Lerer et al. 2019; Zheng et al. 2020; Zhu et al. 2019) try to keep embedding accesses local to each worker; this is done by carefully partitioning the KG across workers and by employing customized negative sampling techniques. The available techniques include *relation partitioning* (Zheng et al. 2020), *graph-cut partitioning* (Zheng et al. 2020), *stratification* (Lerer et al. 2019; Zhu et al. 2019; Mohoney et al. 2021), *shared sampling* (Lerer et al. 2019; Zheng et al. 2020), *local sampling* (Lerer et al. 2019; Zheng et al. 2020; Zhu et al. 2019), and *batch sampling* (Lerer et al. 2019; Zheng et al. 2020). While parallel training methods can handle KGs of such scale and provide reasonable training times, their use may also impact model quality negatively. A comprehensive study of the benefits and drawbacks of these techniques has not been conducted so far.

In this chapter, we report on an extensive experimental study in which we investigated the efficiency and effectiveness of the available parallelization methods. To ensure a fair comparison, we re-implemented all techniques on top of the LibKGE framework (Broscheit et al. 2020). We found that the evaluation methodologies used in prior work are often not comparable and can be misleading in that degradations

in model quality due to parallel training may remain undetected. Our results suggest that current (combinations of) training methods tend to have a negative impact on embedding quality and/or do not provide substantial speedups. We propose a simple but effective variation of the stratification technique used in PyTorch BigGraph that mitigates these problems. We also found that on some datasets (and when combined with suitable sampling techniques) the random-partitioning baseline outperformed more sophisticated methods. Ultimately, we found that efficient and effective parallel training of large-scale KGE models is indeed achievable but requires a careful choice of techniques; the best choice is dataset-dependent. For example, training a large-scale KGE model for Freebase on 8 GPUs is possible with 7× speedup and a model quality that is competitive to sequential methods and exceeds prior results.

Note that following this study, multiple improvements were proposed to the existing parallelization frameworks (Mohoney et al. 2021; Waleffe et al. 2023; Zheng et al. 2024). These improvements mainly address shortcomings of *stratification* partitioning. We expect the presented findings of this study to still hold in combination with newly proposed improvements. In particular, we (partially) addressed multiple of the shortcomings in this study already. For more details on this follow-up work, see Sec. 3.6.1.

3.2 Parallel Training

Typically, KGE models are trained using negative sampling (Sec. 2.4.1). N^- pseudo-negative triples are obtained by corrupting the $N_s^-\times$ the subject and $N_o^-\times$ the object slot of a positive triple. The model is optimized to produce a high score for the positive and low scores for the pseudo-negative triples via gradient-based optimization. This process is performed in mini-batches of size B. For the score calculation of a single triple, the embeddings of the subject and object entity, as well as of the relation are needed. These entity and relation embeddings are the parameters of the model. The key insight that is used by existing parallel training techniques is that when processing a batch, *only the embeddings relevant to the batch need to be accessed and updated*. In particular, the embeddings of the entities and relations of the batch's triples (positives and pseudo-negatives) are relevant, whereas the embeddings of all other entities and relations are not. This insight is exploited to reduce the overhead of parallel processing to the extent possible.

To update the embeddings during training, KGE models rely on optimizers like Adagrad (Duchi et al. 2011) or Adam (Kingma and Ba 2015). These optimizers keep an internal state for each model parameter, i.e., each dimension of an embedding. In

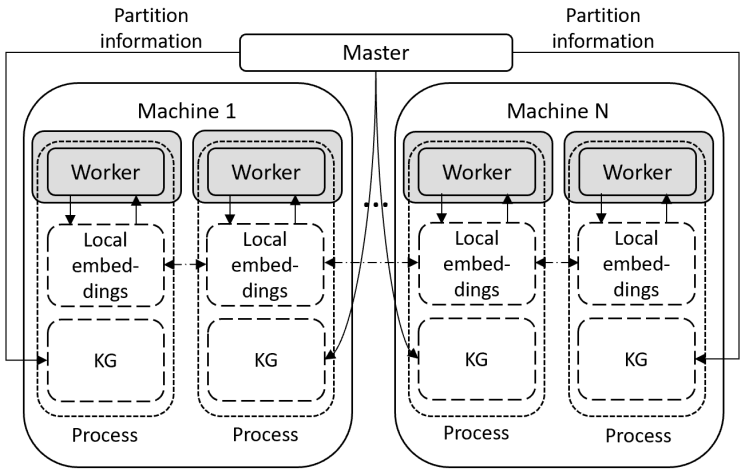


Figure 3.1: General architecture for parallel KGE training. The KG (all triples) is replicated to the workers. A master process indicates workers with the partition information on which part of the triples to work on. Embeddings are distributed across workers using a parameter server. Training is performed on each worker in parallel using a GPU (marked in gray).

the case of Adagrad, this internal state has a space consumption of $O(d(|\mathcal{E}| + |\mathcal{R}|))$. During training, the corresponding state for each processed entity and relation needs to be communicated together with its embeddings. For large models, Lerer et al. (2019) propose to use "row-wise" optimizers that significantly reduce the storage overhead of Adagrad or Adam by maintaining an optimizer state per embedding (instead of per component of each embedding). This reduces the space consumption of the optimizer from $O(d(|\mathcal{E}| + |\mathcal{R}|))$ with Adagrad to $O(|\mathcal{E}| + |\mathcal{R}|)$ with Row-Adagrad.

Here and subsequently, *embedding* refers to the model parameters used to represent a single entity or relation as well as their associated optimizer states.

To facilitate our goal of investigating various techniques to scale KGE training, we make use of a simple architecture (Fig. 3.1) and computational framework (Alg. 3) that generalizes prior multi-GPU and multi-machine KGE training methods and allows to mix and match multiple techniques. For a visualization of this computational framework applied on a specific example, see Fig. 3.2. We first summarize this general framework and subsequently use it to describe the different parallelization methods proposed in the literature.

General architecture. We assume a setup in which multiple workers are coordinated by a master process as in Fig. 3.1. The workers may be situated on a single machine (e.g., for *multi-GPU training*) or distributed across multiple machines. We denote the number of workers by W. The triples $\mathcal{K}$ are replicated to each worker. The master communicates to the worker which subset of the triples it needs to process. We denote such triple subsets as *triple partitions*. Each worker stores a subset of

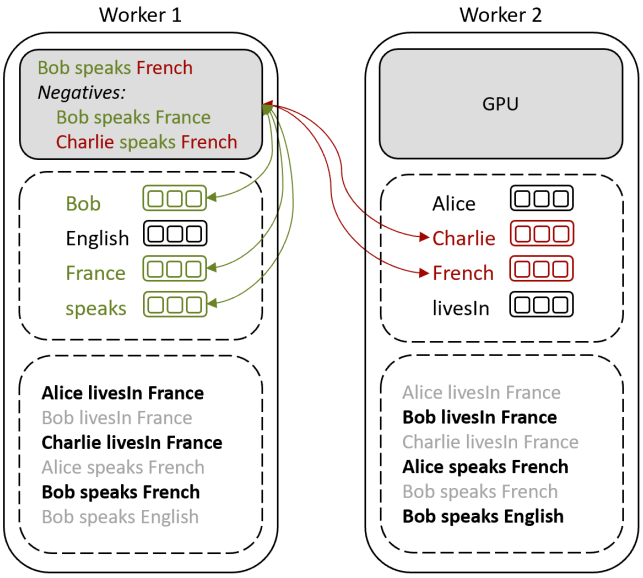


Figure 3.2: Example for processing a single triple of the example KG presented in Fig. 2.2. Triples and embeddings are partitioned via random partitioning (see Sec. 3.3.2) over two workers. Worker 1 performs a training step using the triple (Bob, speaks, French) by (i) creating negative examples (see Sec. 2.4.1), (ii) fetching needed embeddings from local RAM (green) and from remote workers (red) to GPU VRAM (gray), and (iii) pushing update deltas back to local and remote storage locations.

the embeddings of the KGE model. We denote those embedding subsets as *entity* and *relation partitions*; they are defined by the underlying partitioning technique. Synchronization, communication, and updates of embeddings are handled via a co-located parameter server (PS). The PS exchanges data either via shared memory or network.

Master. The general computational framework is described in lines 1–9 of Alg. 3. We start by partitioning the triples of the knowledge graph $\mathcal{K}$ into multiple triple partitions $\mathcal{K}_1, \ldots, \mathcal{K}_P$ (line 2) and initialize the embeddings stored at each worker (line 3). The partitioning technique, the number P of triple partitions as well as the initial location of each embedding depends on the method being used. Training is then performed in multiple epochs until some convergence criterion is met. In each epoch, the master process distributes triple partitions to workers (after optionally repartitioning the data and potentially in a dynamic fashion or in multiple rounds) and waits until all workers have processed their partitions.[a] Note that the cost of distributing triple partitions to workers is generally not a bottleneck: it is done at

[a]The number of triple partitions can be higher than the number of workers. In this case, workers request the next triple partition to process from the master, as soon as they finished processing. The master waits until all triple partitions are processed.

Algorithm 3 Framework for parallel KGE training

Require: A knowledge graph (set of positive triples)
Ensure: KGE model (entity and relation embeddings)

```
1:  MASTER():
2:  Partition knowledge graph                                   ▷ see Sec. 3.3
3:  Initialize embeddings (parallel)
4:  for each epoch do
5:      Repartition triples                                     ▷ optional
6:      Distribute triple partitions to workers
7:      Wait for all workers to complete
8:  end for
9:  return trained embeddings

10: WORKER():
11: while true do
12:     Retrieve triple partition from master                   ▷ latency
13:     Relocate embeddings (async)                             ▷ optional
14:     for each batch do                                       ▷ constructed from current partition
15:         Sample negative examples                            ▷ work; see Sec. 3.4
16:         Pull batch embeddings                               ▷ latency
17:         Process batch on GPU                                ▷ work, latency
18:         Push batch embedding updates                        ▷ latency
19:     end for
20:     Signal completion to master
21: end while
```

once per epoch, and communication cost only involves triple IDs, as all triples are replicated to the workers.

Embedding/parameter management. The embeddings are partitioned across the individual workers and stored in their main memory; the architecture is similar to the key-value store of Zheng et al. (2020). Each worker may *pull* (i.e., read) or *push* (i.e., update) any embedding. The pull operation first retrieves the current value of an embedding from its current location and stores it in GPU memory; likewise, the push operation copies embedding updates from the GPU back to its storage location. Note that the push operation forwards deltas instead of actual values, i.e., the difference between the pulled and updated value; these deltas are applied sequentially. If an embedding is stored locally at a worker, this process is fast, otherwise, latency and communication overheads occur (none if in that worker's GPU memory, some if in that worker's main memory, more if on some other worker). Workers may request to *relocate* an embedding from its current location at a remote worker to themselves to

minimize these overheads.[b] Such relocation allows for efficient implementation of the parameter management schemes used by Zheng et al. (2020) and Lerer et al. (2019) within a common framework. Unless stated otherwise, embeddings are relocated only into the worker's main memory. In some cases (which we will point out explicitly), relocated embeddings are subsequently copied to GPU memory; in this case, the pull and push operations are omitted.[c]

Workers. Each worker repeatedly processes a triple partition as described in lines 10–21 of Alg. 3. After retrieving a triple partition from the master (line 12), each worker may initiate an (asynchronous) relocation of all embeddings relevant to the triple partition to itself, i.e., the entity and relation partitions corresponding to the triple partition. The triple partition is then divided into mini-batches of size B, which are processed in sequence. For each mini-batch, the worker first samples pseudo-negative triples using negative sampling (line 15), then obtains the values of all embeddings relevant for the batch (line 16), processes the batch on a GPU (line 17), and (asynchronously) writes back the update deltas to the PS (line 18). After the entire triple partition has been processed, the worker signals completion to the master.

Concurrent embedding accesses. During training, some embeddings may be concurrently accessed by multiple workers, leading to conflicts. Due to the sparsity of KGs, the number of conflicts is low for most embeddings (Zhang et al. 2017), but they may arise more frequently for relation embeddings and highly connected entities. Some partitioning techniques (Sec. 3.3) specifically minimize these access conflicts, but they generally cannot be completely avoided. In such cases, access to stale embedding versions may occur. Our PS ensures sequential consistency, however. Moreover, lost updates do not occur because the push operation forwards deltas instead of actual values, and these deltas are applied sequentially.

Discussion. In Alg. 3, we marked each step of the worker with whether (i) actual work is performed and (ii) latency due to data movement may arise. Note that only negative sampling (line 15) and batch processing (line 17) of our framework correspond to "training work"; the rest of the framework constitutes coordination and communication overhead. Key to effective parallel training is to reduce this overhead to the extent possible (most notably, in lines 16–18), e.g., by reducing the amount

[b] In our implementation, we use the Lapse parameter server (Renz-Wieland et al. 2020), which supports transparent parameter relocation (see Sec. 3.5.2).

[c] Embeddings are relocated to a GPU only if (i) there is sufficient memory on the GPU and (ii) it can be ensured that no other worker accesses these embeddings. In this case, the updated embeddings are copied back from the GPU to the worker's main memory once the entire triple partition has been processed (line 20 in Alg. 3).

		PBG (Lerer et al. 2019)		DGL-KE (Zheng et al. 2020)		GraphVite (Zhu et al. 2019)		This study	
		1	D	1	D	1	D	1	D
Partitioning	Random						N/A	✓	✓
	Relation			✓			N/A	✓	✓
	Graph-Cut				✓		N/A	✓	✓
	Stratification	✓	✓			✓	N/A	✓	✓
Neg. samp.	Uniform						N/A	✓	✓
	Shared	✓	✓	✓	✓		N/A	✓	✓
	Local	✓	✓		✓	✓	N/A	✓	✓
	Batch	✓	✓	✓	✓		N/A	✓	✓

Table 3.1: Summary of techniques for parallel KGE training. 1 refers to single-machine multi-GPU setup, *D* means multiple machines.

of data that is communicated between workers and between main memory and GPU memory as well as by minimizing the impact of communication latency.

Techniques. Pytorch BigGraph (PBG) (Lerer et al. 2019), GraphVite (Zhu et al. 2019), and DGL-KE (Zheng et al. 2020) provide methods for scalable training of KGE models. The main goals of these methods are (i) low communication overhead, (ii) fast epoch times, and (iii) high embedding quality. Generally, careful partitioning and embedding relocation as well as customized negative sampling strategies are required to obtain effective methods, and the frameworks differ mainly w.r.t. how this is done. We discuss partitioning in Sec. 3.3 and negative sampling in Sec. 3.4. An overview of the various techniques is given in Tab. 3.1.

3.3 Partitioning

We are now ready to describe and compare various approaches to partition the KG across workers (lines 2 and 5 of Alg. 3). The most basic partitioning scheme, which serves as a baseline for our study, is random partitioning. We also describe relation partitioning, graph-cut partitioning, and finally stratification and discuss their influence on the performance of parallel training. An overview of which framework proposed or used which partitioning method is given in Tab. 3.1. Following this study, further frameworks have been proposed, mainly improving on techniques presented with PBG. For an overview, see Sec. 3.6.1.

	Load balancing	Variety	Comm. cost
Random	+	+	-
Relation	◦	+	◦
Graph-cut	-	-	+
Stratification (plain)	◦	-	+
Stratification (CARL)	◦	◦	+

Table 3.2: Influence of partition approaches on balancing of partition sizes, variety, and communication cost.

3.3.1 Desiderata for Partitioning Techniques

The choice of the partitioning method influences all three goals of parallel training mentioned above: *low communication cost*, *fast epoch times*, and *high quality*. We discuss each goal in turn; see Tab. 3.2 for a coarse overview of the influence of each partitioning scheme on these three criteria.

Low communication cost. First, the choice of partitioning scheme influences communication cost because it determines which data is processed by which worker and consequently which embeddings are accessed by which workers. If the partitioning of the embeddings across workers is well-coordinated with the partitioning of the triples, latency can be largely avoided and overall communication cost reduced. Generally, this is done by allocating embeddings to the workers that access them frequently (line 13 of Alg. 3). Note that the communication overhead can further be reduced by selective replication of embeddings, as done in follow-up work (Renz-Wieland et al. 2023). For more insights on such improvements, see Sec. 3.6.2.

Fast epoch times. Next, assuming that communication latency has been addressed and is not a bottleneck, the wall-clock time taken for an epoch is mainly driven by *load balancing*: If each worker has a similar amount of work, all workers can operate in parallel. Otherwise, overloaded workers may stall progress and become a bottleneck (e.g., line 7 in Alg. 3). To assess whether a partitioning scheme balances computation, we use triple partition sizes as a proxy: We generally prefer partitionings in which triple partition sizes are balanced over partitionings in which they are not. Another factor is the number P of partitions: many small partitions ($P \gg W$) may induce higher cost than a few larger partitions (e.g., $P = W$), mainly due to overheads incurred when switching partitions at a worker.

High Quality. Finally, and perhaps less obviously, the partitioning scheme also influences the quality of the resulting embeddings. On the one hand, this is because some partitioning schemes limit *"variety"* in that the distribution of co-occurrence of some triples (or entities) is changed so that they do not or only seldom co-occur

in a partition or a batch. On the other hand, and perhaps more importantly, the partitioning schemes also influence the impact on quality of the parallel negative sampling techniques in that the negative sampling pool might be dependent on the entity partitions. These influences will be discussed in Sec. 3.4.

3.3.2 Random Partitioning

In random partitioning, the underlying triples $\mathcal{K}$ of the KG $\mathcal{G}$ are randomly divided into P equally-sized triple partitions $\mathcal{K}_1, \ldots, \mathcal{K}_P$, where typically P equals the number of workers W. Likewise, all embeddings are partitioned randomly across workers, resulting in P entity/relation partitions. An example training step with random partitioning is visualized in Fig. 3.2.

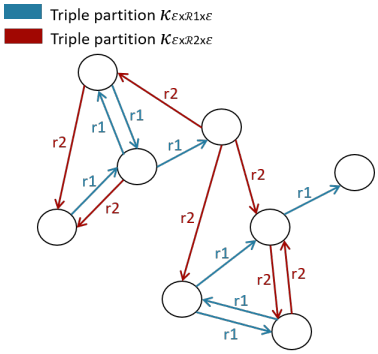
Discussion. Random partitioning ensures perfect load balancing and wide variety of triples within each triple partition. Its main problem is communication cost. Since the partitioning ignores the structure of the knowledge graph, most embedding accesses (pull and push in Alg. 3) will be non-local and incur communication and latency: each individual embedding access is local with a probability of only $1/W$. The resulting cost of remote embedding accesses may exceed potential parallelization benefits.

3.3.3 Relation Partitioning

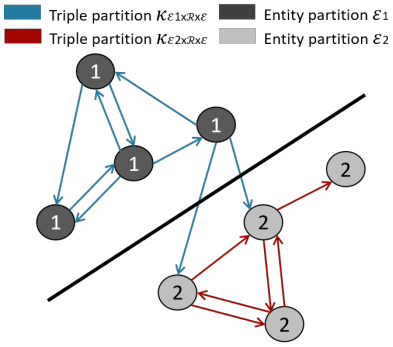
The random partitioning scheme can be improved by making use of the following observation. In most large KGs, the number of entities (say, multiple millions) exceeds by far the number of relations (say, a few thousand). Since the processing of each SPO triple in a batch requires access to its corresponding entity and relation embeddings, a substantial fraction of the embedding accesses are to relation embeddings. In addition to being responsible for a substantial amount of communication (O(1/3)) in random partitioning, there are generally more conflicts between these accesses in that multiple workers may access given relation embeddings simultaneously (just because there are few relations).

Both problems can be avoided by relation partitioning (Zheng et al. 2020). Here the set $\mathcal{R}$ of relations is partitioned into P subsets $\mathcal{R}_1, \ldots, \mathcal{R}_P$, where as before $P = W$. The KG is then partitioned accordingly: triple partition $\mathcal{K}_p$ consists of the triples with a relation in $\mathcal{R}_p$, i.e., $\mathcal{K}_p = \mathcal{K} \cap \mathcal{E} \times \mathcal{R}_p \times \mathcal{E}$. An example of relation partitioning with $P = 2$, $\mathcal{R} = \{ r_1, r_2 \}$ and $\mathcal{R}_p = \{ r_p \}$ is shown in Fig. 3.3a.

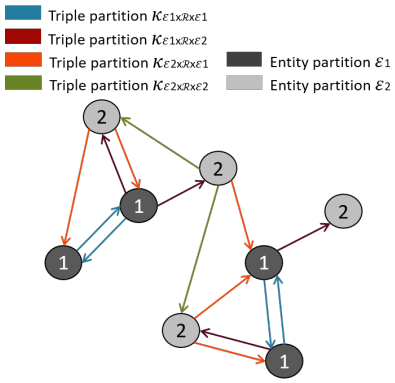
Discussion. In general, relation partitioning aims to provide balanced triple partition sizes. Finding such a partition is a *multiway number partitioning* problem (Graham 1966) (where the numbers correspond to relation sizes) and greedy approximation



(a) Relation partitioning: Triple partition membership is determined by each triple's relation (here: r_1 and r_2).



(b) Graph-cut partitioning: Triple partition membership is determined by each triple's subject entity. Entities are partitioned via a graph cut (indicated by vertex color).



(c) Stratification partitioning: Triple partition membership is determined by each triple's subject and object entity. Entities are partitioned randomly (indicated by vertex color).

Figure 3.3: Illustration of partitioning approaches for $W = 2$ workers. Colors indicate triple partition membership.

methods[d] empirically tend to work well as long as there are no overly large relations. However, such relations do arise in practice; e.g., the most frequent relation occurs in between 3% and 35% of the training triples of the datasets used in this study. In such cases, triple partitions either become unbalanced (impacting load balancing) or the relation partitioning is "softened" by splitting the triples of large relations across workers (impacting communication cost).

Since in (hard) relation partitioning each relation occurs only at a single worker, relation embeddings do not need to be communicated across workers and can, in fact, be stored in each worker's GPU memory during the relocation step (line 13 in

[d] We use the popular heuristic of sorting the relations in descending order first, before applying greedy-number-partitioning.

Alg. 3); no subsequent costs arise for pulling or pushing these embeddings. If soft relation partitioning is used, this benefit vanishes for frequent relations. There is also no benefit for accessing entity embeddings so substantial communication costs may still arise. These costs may be slightly reduced by relocating each entity to the worker that accesses it most often; this is especially beneficial if certain entities only arise in a single relation.

Finally, relation partitioning reduces variety when compared to random partitioning. Whether two triples may co-occur in a triple partition (and consequently a batch) depends on the relations of these triples.

3.3.4 Graph-Cut Partitioning

An alternative to partitioning the relations is to partition the set $\mathcal{E}$ of entities into P entity partitions $\mathcal{E}_1, \ldots, \mathcal{E}_P$, which are distributed across the workers. The triples $\mathcal{K}$ are partitioned accordingly in that each triple is assigned to a triple partition that corresponds to the entity partition of its subject entity: We have $\mathcal{K}_p = \mathcal{K} \cap \mathcal{E}_p \times \mathcal{R} \times \mathcal{E}$ for $1 \leq p \leq P$. Triple partition $\mathcal{K}_p$ is processed by the worker at which $\mathcal{E}_p$ is located. The idea of graph-cut partitioning (Zheng et al. 2020) is to partition the entities in such a way that (i) partition sizes $|\mathcal{K}_p|$ are balanced and (ii) most triples are local to their partition. A triple $(s, r, o) \in \mathcal{K}_p$ is *local* if $s, o \in \mathcal{E}_p$: in this case, no communication is required to access the corresponding entity embeddings during training. Zheng et al. (2020) use METIS (Karypis and Kumar 1998) to create such a balanced graph cut; see Fig. 3.3b for an example.

Discussion. Entity partitions influence the communication cost between workers and triple partitions the computational cost at each worker. There is generally a trade-off between balancing the sizes of entity and triple partitions and the fraction of local triples. The communication overhead for relation embeddings is not reduced. Most knowledge graphs admit a well-balanced cut in terms of entity partitions, but these cuts result in unevenly-sized triple partitions; e.g., our experimental study showed a coefficient of variation of triple partition sizes of up to 37%, leading up to 2× longer processing times of the largest partition compared to the smallest (c.f. Tab. 3.16).

As the effectiveness in terms of training time reduction is heavily dependent on this balancing, graph-cut partitioning is beneficial for datasets for which there is a good graph cut with balanced entity partitions *and* balanced triple partitions. In general, many entities occur in multiple partitions so data movement across workers as well as between main memory and GPU memory cannot be avoided. Finally, entity partitions tend to contain highly interconnected entities (*communities*), which

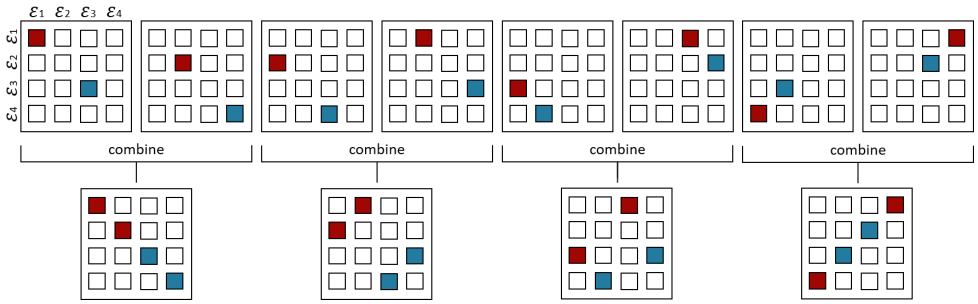


Figure 3.4: Possible schedule for $W = 2$ workers with stratification using $M = 4$ entity partitions. Each block indicates one scheduling step processing one stratum. Each color indicates one worker and each marked triple partition inside a stratum is processed in parallel by a worker. Combined stratification (bottom) halves the number of strata compared to plain stratification (top) by combining mirror partitions.

substantially reduces variety across partitions. This is especially problematic in conjunction with some of the sampling techniques discussed in Sec. 3.4.

3.3.5 Stratification Partitioning

Stratification partitioning was originally introduced in the context of matrix factorization (Gemulla et al. 2011) and subsequently adapted to training KGEs (Lerer et al. 2019; Zhu et al. 2019). The idea behind stratification is to create triple partitions (termed *buckets* in (Lerer et al. 2019)) that access *pairwise disjoint sets of entities*. A set of S such triple partitions is called a *stratum* (Gemulla et al. 2011). The key insight behind stratification is that we can process the triple partitions within a stratum in parallel across S workers without any need to synchronize entity embeddings in between as they are pairwise disjoint.[e] Strata can be obtained in the following way: first partition the set $\mathcal{E}$ of entities randomly into M entity partitions $\mathcal{E}_1, \dots, \mathcal{E}_M$ (where generally $M = \Theta(W)$) and subsequently partition the triples based on their subject *and object* entities. For example, a triple with a subject of entity partition p_1 and an object of entity partition p_2 is assigned to the triple partition $\mathcal{K}_{p_1 p_2}$. We obtain $P = M^2$ triple partitions of form $\mathcal{K}_{p_s p_o} = \mathcal{K} \cap (\mathcal{E}_{p_s} \times \mathcal{R} \times \mathcal{E}_{p_o})$ for $1 \leq p_s, p_o \leq M$. Such a partitioning is visualized in Fig. 3.3c for $M = 2$. The resulting triple partitions are then scheduled across workers such that at any point in time, the set of currently processed triple partitions forms a stratum (see below). An example schedule for $M = 4$ and $W = 2$ is shown in Fig. 3.4 (top). Before processing a triple partition,

[e] Note that synchronization may be required to process negative samples. We come back to this point in Sec. 3.4.

workers relocate their entity embeddings to themselves (line 13 of Alg. 3) so that accesses to these entities are local.[f]

Partitioning and scheduling. The number M of entity partitions must be carefully chosen. Generally, any two triple partitions $\mathcal{K}_{p_s p_o}$ and $\mathcal{K}_{p'_s p'_o}$ within a stratum must be guaranteed to contain disjoint entities, i.e., we require $\{ p_s, p_o \} \cap \{ p'_s, p'_o \} = \emptyset$. A natural choice is to set $M = W$; however, such a choice does not ensure that all strata contain W triple partitions and thus saturate all workers. In Fig. 3.3c, where $M = 2$, it is only possible to build the non-trivial stratum $\{ \mathcal{K}_{11}, \mathcal{K}_{22} \}$; all other pairs of triple partitions do not form a stratum and thus cannot be processed in parallel. During training, this means that $\mathcal{K}_{12}$ and $\mathcal{K}_{21}$ are processed sequentially and one worker remains idle. We call such a schedule *blocking*.

To increase worker saturation, Lerer et al. (2019) proposed to set $M > W$; in particular $M = 2W$ (less does not suffice). The resulting $P = (2W)^2$ triple partitions can be grouped into $4W$ strata, each of size exactly W. An example of such a non-blocking schedule is shown in Fig. 3.4 for $W = 2$ and consequently $M = 4$. Scheduling must be done carefully. A best-effort scheduler such as the one used in PBG (Lerer et al. 2019) does not always produce a non-blocking schedule. Follow-up work improved on this scheduling for single (Mohoney et al. 2021; Waleffe et al. 2023) and multi-GPU (Zheng et al. 2024) training. For an overview of improvements, see Sec. 3.6.1.

Processing order. Processing a triple partition only leads to updates in its corresponding entity partitions. For this reason, the embedding spaces of each entity partition may not be aligned in the first epochs when trained with a single worker. This problem can be mitigated by a carefully chosen processing order (Lerer et al. 2019). In our setting, this issue vanishes as concurrent updates of relation embeddings by multiple workers automatically lead to aligned embedding spaces. In fact, a random ordering of strata performed best in our experiments.

Discussion. Since entities are partitioned randomly, all triple partitions have the same size in expectation ($= N/P = N/M^2$). Triple partition sizes may still vary somewhat; the variance depends on the data distribution. A key problem with stratification is that the number of triple partitions scales quadratically with the number of workers (since $M = \Theta(W)$). This is problematic especially when the KG is small relative to the number of workers. The main advantage of stratification is that (i) overall communication cost is reduced and (ii) no access latency arises during batch

[f] Relocation is to main memory in general to allow across-partition negative sampling. When local sampling is used (see Sec. 3.4.4), relocation to GPU is possible.

processing for the embeddings of the partition's entities. Assuming a non-blocking schedule and ignoring negative sampling, each entity embedding is relocated at most P/W times (e.g., $4W$ times for the choice of $M = 2W$). In practice, roughly half of the relocations can be avoided by scheduling triple partitions to workers that already have their subject or object entities localized. Finally, stratification does reduce variety between triple partitions, but it does so in a less systematic way than graph-cut partitioning.

3.3.6 Improved Stratification Partitioning (CAR)

Stratification partitioning is a promising technique but, as discussed above, may suffer from (i) small triple partitions, (ii) many relocations, and (iii) reduced variety. We propose three simple techniques that mitigate these problems: (i) combining mirror partitions (C), (ii) relocating only active entities (A), and (iii) repartitioning between epochs (R). We refer to the resulting variant of stratification as *CAR stratification* and study its impact in the experimental section.

Combining mirror partitions (C) is a technique that halves the number of triple partitions required to create a non-blocking schedule. It consequently reduces communication cost and leads to more variety within triple partitions. The approach works as follows: First compute triple partitions as in plain stratification and subsequently merge each pair of *mirror partitions* into a single triple partition. The mirror partition of $\mathcal{K}_{p_s p_o}$ is $\mathcal{K}_{p_o p_s}$ when $p_s \neq p_o$. When $p_s = p_o = p$, the mirror partition is $\mathcal{K}_{(p+1)(p+1)}$ for odd p and $\mathcal{K}_{(p-1)(p-1)}$ for even p. An example is shown in Fig. 3.4 (bottom); here, mirror partitions are indicated by the same color. One can show that when $M = 2W$, mirror partitions also admit a non-blocking schedule such as the one shown in Fig. 3.4 (bottom). Roughly speaking, when some worker w processes $\mathcal{K}_{p_s p_o}$, plain stratification ensures that no other worker processes mirror partition $\mathcal{K}_{p_o p_s}$ concurrently. We can thus merge each partition with its mirror.

Active entities (A). In plain stratification, the entity embeddings of the combined entity partitions $\mathcal{E}_{p_s} \cup \mathcal{E}_{p_o}$ are relocated to the worker before processing triple partition $\mathcal{K}_{p_s p_o}$. However, only a fraction of these entities may actually occur either as subject or object of a triple in $\mathcal{K}_{p_s p_o}$. We denote those entities as *active* and propose a simple variant that relocates only the embeddings of active entities instead. This approach reduces communication cost significantly when partitions are sparse (as is often the case in large KGs). Furthermore, this affects the pool for the local sampling technique discussed in Sec. 3.4.4.

Batch (pos. triples)	Uniform negatives	Shared negatives	Batch negatives
(Hitchcock, directed, Psycho)	Computer, LionKing	Computer, LionKing	GreenBook, Ali
(Ali, ActorIn, GreenBook)	University, Biden	Computer, LionKing	Ali, Psycho

Table 3.3: Example for negative samples produced by various techniques ($B = 2$, $N^- = 2$). Only the sampled corrupted entities are shown. In *shared sampling*, corrupted triples are reused across positive triples. In *batch sampling*, corrupted entities are sampled from the positive triples occurring in the batch.

Repartitioning (R). The entity partitioning and hence the triple partitioning in plain stratification is static, i.e., it does not change between epochs. To increase variety, one may repartition both entities (randomly) and triples between epochs. This repartitioning can be performed in the background and only incurs minor communication costs when each worker has access to the complete knowledge graph. Note that repartitioning is also possible for random partitioning, but generally not for graph-cut and relation partitioning.

Discussion. All of the above modifications are simple. Our experimental study suggests that they are effective or even instrumental for the efficient use of stratification.

3.4 Negative Sampling

The partitioning techniques discussed in the previous section aim to make access to (the entity and relation embeddings of) positive triples from $\mathcal{K}$ more efficient. We now turn attention to pseudo-negative triples. Recall that these triples are constructed by negative sampling, i.e., for each positive triple in a batch, we create N^- pseudo-negative triples by corrupting either its subject or its object with a randomly sampled entity. All embeddings of the so-sampled entities needed to be pulled to the worker as well, which induces communication overhead. If N^- is large, this overhead is substantial and may by far outweigh the communication overhead required to process positive triples.

The communication overhead is mainly driven by (i) the number of unique entities sampled for a batch[g] and (ii) the location of the corresponding entity embeddings. Both factors can be influenced by biasing the sampling distribution used for corrup-

[g] If an entity occurs multiple times in a batch, its embedding needs to be pulled/pushed only once.

Technique	Unique entities per batch
Uniform sampling	$2B + BN^-$
Shared sampling	$2B + N^-$
Batch sampling	$2B$

Table 3.4: Number of unique entities per batch (upper bound) by sampling technique. Corresponding entity embeddings need to be communicated and copied to GPU memory.

tion. The key techniques are *shared sampling* (Lerer et al. 2019; Zheng et al. 2020), *batch sampling* (Lerer et al. 2019; Zheng et al. 2020), and *local sampling* (Lerer et al. 2019; Zheng et al. 2020; Zhu et al. 2019). An example per technique is given in Tab. 3.3, and an overview of the techniques used in various frameworks in Tab. 3.1. In the following, we describe the techniques and analyze them in terms of the number of unique entities being used (Tab. 3.4).

3.4.1 Uniform Sampling

Uniform sampling is the basic technique used in sequential methods for training KGE embeddings; we use it as a baseline. Each negative sample is obtained by sampling the corrupted entity uniformly and independently from the set $\mathcal{E}$ of all entities. This leads to a large number of unique entities per batch: with a batch size of B, up to $2B$ unique entities occur in the positive triples (subject and object entity) and up to BN^- additional unique entities in the pseudo-negative triples (corrupted entity). Consequently, even for small choices of N^-, most entities relevant for a batch arise from negative sampling. In expectation, only a fraction of $1/W$ of these entities is local to the worker so communication costs are high.

3.4.2 Shared Sampling

The idea of *shared sampling* (Lerer et al. 2019; Zheng et al. 2020) is to use the same corrupted entities for triples in a batch. In more detail, N^- triples are sampled uniformly and independently from the set of all entities and then shared across positive triples.[h] This substantially reduces the number of unique corrupted entities from BN^- to N^- and consequently leads to a significant reduction of communication costs and epoch time. For many KGE models, computational cost can also be reduced because it facilitates the scoring of all negative triples in a batch on the GPU via matrix multiplication. This approach is used in PBG and DGL-KE. A key drawback of shared sampling is that variety is reduced, which can have a negative impact on

[h] Shared sampling can also be applied to a subset of the batch as done by Zheng et al. (2020). However, we did not see any benefits of this approach.

embedding quality. In fact, our experimental study (Sec. 3.5.5) suggests that (i) this issue can be mitigated by increasing the number of negatives drastically, and (ii) even with the higher number of negative samples, shared sampling is far more efficient than uniform sampling.

3.4.3 Batch Sampling (B)

To avoid transferring additional embeddings for the entities of negative triples (both to worker and to GPU memory), *batch sampling* (termed degree-based sampling in (Zheng et al. 2020)) does not sample from the set $\mathcal{E}$ of all entities but from the set of entities that already occur in a positive triple of the batch. The number of unique entities per batch reduces to $2B$ and in particular, does not depend on the number N^- of negative samples anymore. The size of the sampling pool for the negative samples is heavily reduced and their distribution is data-dependent (more frequent entities are more likely to occur in a batch and thus to occur as a negative). Zheng et al. (2020) combine batch sampling with uniform sampling to increase variance. Our experimental study suggests that the benefits of batch sampling depend on both dataset and partitioning technique. On most datasets, batch sampling heavily deteriorated embedding quality, but it was highly beneficial in terms of both embedding quality and communication overhead on the Freebase dataset. Here, uniform negatives may lead to "easy" negative samples (Kotnis and Nastase 2017), whereas batch samples may contain harder negatives.

3.4.4 Local Sampling (L)

Another approach to bias the sampling distribution is *local sampling*, where corrupted entities are drawn from the set of entities whose embeddings are located at the worker (in the parameter server). This ensures that pull operations for negatives do not incur communication with other workers. Local sampling is used by DGL-KE, PBG, and GraphVite and can be combined with shared sampling.

Since local sampling biases the sampling distribution, it has an impact on embedding quality. This impact is mainly determined by the allocation of entity embeddings across workers, which in turn is driven by the partitioning scheme being used. For example, if graph-cut partitioning is used, the allocation is static: for all triples of partition $\mathcal{K}_p$, negative samples are taken from $\mathcal{E}_p$. Likewise, when processing partition $\mathcal{K}_{p_s p_o}$ in stratification partitioning, negative samples are taken from the set $\mathcal{E}_{p_s} \cup \mathcal{E}_{p_o}$ of entities (or the subset of active entities in CAR stratification). Con-

sequently, local sampling may degrade quality in all these cases; we investigate this degradation empirically in our experimental study.

One counter-measure to mitigate degradation effects is to repartition the data between epochs as in CAR stratification. Such an approach is not possible for graph-cut partitioning (since partitions are deterministic), however, and it is very restricted in relation partitioning. An alternative approach that ensures a dynamic local-sampling pool is to reshuffle the entity partitions (but not the triple partitions) randomly between epochs. This approach—denoted (r)—can be used with random and relation partitioning since these methods use a random entity allocation.

Note that when local sampling is combined with stratification partitioning, it is guaranteed that (i) all entity embedding accesses (positive and negative) of a worker are local and (ii) no other worker accesses these embeddings concurrently. For this reason, the embeddings can be stored directly in GPU memory, and the pull/push operations (lines 16 and 18 of Alg. 3) can be omitted.

3.5 Experimental Study

We conducted an experimental study in which we investigated the various partitioning and negative sampling techniques in a common framework. Our main goal was to provide insight into (i) the efficiency of the techniques in terms of runtime as well as, (ii) their effectiveness in terms of embedding quality, and ultimately (iii) whether and to what extent parallelization is beneficial to train KGE models. We consider both single-machine multi-GPU scenarios as well as multi-machine multi-GPU training.

3.5.1 Key Findings

Before describing the experimental setup and results of our study in detail, we briefly summarize our key findings.

General findings.

1) The right choice of partitioning and negative sampling technique was crucial. With such a choice, both multi-GPU and multi-machine parallelization were effective and efficient, especially for larger knowledge graphs (Sec. 3.5.3).
2) The right choice is dataset-dependent.
3) With the right choice, 7× speedup with 8 GPUs was possible.
4) The sampled MRR metric (Sec. 2.5) used in prior work to evaluate KGE model quality on large KGEs is misleading and should not be used (Sec 3.5.4).

5) Row-based optimizers such as Row-Adagrad reduced communication overhead while preserving a high embedding quality (Sec. 3.5.9).
6) The findings are stable over varying configurations of hyperparameters (Sec. 3.5.10).

Partitioning techniques (Sec. 3.5.3).

1) CAR stratification in combination with local sampling was most efficient and effective on most datasets. It also outperformed plain stratification.
2) Random partitioning consistently led to high-quality embeddings. On Freebase, it was the overall method of choice.
3) Graph-cut partitioning was least effective due to unbalanced triple partition sizes (Sec. 3.5.7).
4) Relation partitioning led to time improvements but had a negative influence on embedding quality on some datasets.

Sampling techniques (Sec. 3.5.5, 3.5.6).

1) Shared sampling was instrumental for efficiency and should always be used.
2) Local sampling with a dynamic sampling pool improved efficiency and should be used on large graphs.
3) A static local sampling pool often had a negative influence on embedding quality.
4) Batch sampling significantly deteriorated resulting quality on most graphs. On Freebase, however, it significantly improved quality.

3.5.2 Experimental Setup

We provide code, search configurations, and resulting hyperparameters at
`https://github.com/uma-pi1/dist-kge`.

Datasets. We used knowledge graphs of varying sizes; see Tab. 3.5. *FB15K* (Bordes et al. 2013), a small subset to compare to prior work on KGE, *Yago3-10* (Dettmers et al. 2018), *Wikidata5M* (Wang et al. 2021b) and the *Freebase* KG as used by Zheng et al. (2020) and Lerer et al. (2019).[i] Note that a KGE model for Freebase does not fit on a single GPU (e.g., a ComplEx model with $d = 400$ takes 128 GB plus optimizer state). For all datasets except Freebase, we use the validation and test sets that accompany the datasets. For Freebase, we sample 10 000 triples from the original validation and test sets (which contain about 17M triples each) to keep evaluation costs feasible. These sets are still sufficiently large to estimate model quality accurately.

[i] dump: `https://developers.google.com/freebase/`, preprocessed: `http://web.informatik.uni-mannheim.de/pi1/kge-datasets/freebase.tar.gz`.

Dataset	Entities	Relations	|Train|	|Valid|	|Test|
FB15K	14 951	1 345	483 142	50 000	59 071
Yago3-10	123 182	37	1 079 040	5 000	5 000
Wikidata5M	4 818 579	828	21 343 681	5 357	5 321
Freebase	86 054 151	14 824	304 727 650	16 929 318	16 929 308

Table 3.5: Datasets used in this study.

Hardware. We ran all experiments on the same hardware. We used two machines with 40 CPUs (Intel(R) Xeon(R) CPU E5-2640 v4, 2.40GHz) and 4 GPUs (GeForce RTX 2080 Ti). The bandwidth between the machines was 1 GB/s. For each experiment, we write $G@C$ to indicate that G GPUs were used on each of C machines. For example, a 1@2 experiment uses two machines with one GPU each. On each GPU, we run two workers so that the 1@2 setup trains with four workers.

Implementation. We implemented parallel training on top of the PyTorch-based LibKGE library (Broscheit et al. 2020), which provides a number of KGE models, training methods, and evaluation techniques. KGE models can be grouped into semantic matching models and translational distance models (Sec. 2.3.1). We considered ComplEx (Trouillon et al. 2016) and RotatE (Sun et al. 2019), which are among the currently best-performing models (Ali et al. 2021a; Sun et al. 2019; Ruffinelli et al. 2020; Lacroix et al. 2018) for the former and latter group, respectively. For single-machine multi-GPU training, we used a shared-memory PyTorch tensor to store embeddings in main memory. For multi-machine training, we used the parameter server Lapse (Renz-Wieland et al. 2020).

Metrics. We compared parallel to sequential training methods. We used two different sequential settings: *sequential (GPU memory)* and *sequential (main memory)*, which differ in the location of the embeddings. For the main memory version, embeddings are copied to and retrieved from the GPU memory when processing each batch; this allows for handling very large KGE models. We evaluated efficiency in terms of runtime and communication cost (GBs transferred) per epoch. To evaluate quality, we follow standard practice and compute the filtered mean reciprocal rank (MRR) and Hits@k for the link prediction task; for details see Sec. 2.5. Finally, to evaluate effectiveness, we combined efficiency and result quality, which may be affected by parallelization. In particular, for each setting, we report the time required to reach an MRR that exceeds 95% of the best MRR achieved in the sequential setting.

Hyperparameter optimization. A solid choice of hyperparameters is key to obtaining high-quality KGE models. In general, we followed Ruffinelli et al. (2020) and performed the model selection using a quasi-random search with 30 trials in a sequen-

B	Batch sampling	Sec. 3.4.3
L	Local sampling	Sec. 3.4.4
A	Sampling from active entities only	Sec. 3.3.6
r	Reshuffle entity partitions every epoch	Sec. 3.4.4
R	Repartition triple and entity partitions every epoch	Sec. 3.3.6
C	Combining mirror partitions	Sec. 3.3.6

Table 3.6: Summary of short notations.

tial setup. Each trial was run for 20 epochs using Adagrad; the best resulting model was then run for up to 300 epochs (Wikidata5M) or 400 epochs (FB15K, Yago3-10). An exception is Freebase; (which would be infeasible to run sequentially); here we followed Lerer et al. (2019) and used the best-performing setting on FB15K, and trained up to 10 epochs using Row-Adagrad. Evaluating the effects of this hyperparameter selection based on a smaller graph, we additionally used the large-scale hyperparameter optimization approach GraSH (Ch. 4) and compared findings (Sec. 3.5.10). Note that results on Freebase tuned on FB15K are considerably behind performance using the properly tuned hyperparameter configurations. However, general findings hold over both sets of hyperparameters. For our parallel experiments, we generally used the best-performing sequential hyperparameters. We used 128-dimensional embeddings for all datasets.

Techniques. As shared negative sampling was instrumental for efficient parallel training, we consistently used it unless noted otherwise. We explored local (L) and batch (B) sampling, repartitioning of triple partitions and relocation of corresponding entity partitions (R), shuffling of entity partitions (r), combined mirror triple partitions (C), and active entities (A). For an overview of short notations see Tab. 3.6. We use batch sampling combined with uniform sampling (50/50) unless stated otherwise. For stratification we set $M = 2W$ for all datasets but Freebase. For Freebase, we used $M = 32$ throughout so that the entity partitions fit in GPU memory in all settings.

3.5.3 Partitioning Techniques (Tab. 3.7–3.10)

Tab. 3.7 (ComplEx) and Tab. 3.8 (RotatE) summarize the most effective parallelization techniques for various numbers of GPUs and machines. For all settings but 4@2, we ran all partitioning techniques and reported the one with the lowest time to 0.95 MRR. For 4@2, we used the best-performing partitioning technique found for 1@2 (2@2 for Freebase). Parallelization was effective on all datasets, but the speedups on Yago3-10 were small; here parallelization overheads and quality degradation dominated. For the larger Wikidata5M and Freebase datasets, speedups were significant: up to 7× without

	Set up	Partitioning technique	Epoch time	Time to 0.95 MRR	MRR	Hits @10
FB15K	1@1	Seq. (GPU memory)	5.9s	3.9min	**0.779**	**0.862**
	1@1	Seq. (main memory)	7.7s	5.1min	**0.779**	**0.862**
	2@1	Random (R)	2.6s	2.0min	0.775	0.859
	1@2	Random (R)	2.9s	2.2min	0.775	0.859
	4@2	Random (R)	**1.3s**	**1.3min**	0.766	0.858
Yago3-10	1@1	Seq. (GPU memory)	24.3s	38.5min	0.542	0.675
	1@1	Seq. (main memory)	42.6s	67.5min	0.542	0.675
	2@1	Relation	19.0s	**33.2min**	0.538	0.669
	1@2	Random (RL)	19.5s	35.8min	**0.547**	**0.679**
	4@2	Random (RL)	**5.6s**	n.r.	0.503	0.653
Wikidata5M	1@1	Seq. (GPU memory)	438.4s	219.0min	0.297	0.385
	1@1	Seq. (main memory)	774.3s	387.0min	0.297	0.385
	2@1	Stratification (CARL)	232.8s	77.6min	**0.308**	**0.398**
	1@2	Stratification (CARL)	228.0s	**76.0min**	**0.308**	**0.398**
	4@2	Stratification (CARL)	**97.1s**	105.2min	0.294	0.377
Freebase	1@1	Seq. (main memory)	3929.0s	-	0.364	0.487
	4@1	Random (RLB)	704.6s	-	**0.426**	**0.529**
	2@2	Random (RLB)	966.7s	-	**0.426**	**0.529**
	4@2	Random (RLB)	**591.6s**	-	0.421	0.523

Table 3.7: Partitioning techniques (best-performing variant in terms of time to 0.95 MRR). Calculating time to 0.95 MRR was too expensive for Freebase. (ComplEx, n.r. means not reached)

any quality degradation. The two models behaved similarly but overall RotaE training times were higher compared to ComplEx. Furthermore, graph-cut partitioning had a smaller negative influence on quality with RotatE than with ComplEx.

Overall, the best-performing techniques were dataset-dependent. In most cases, a variant of stratification or random partitioning outperformed other techniques or was close to the best result. We analyzed Freebase separately (Tab. 3.10). Here the random partitioning baseline in combination with local (L) and batch (B) sampling was most effective; stratification led to the overall worst results. One reason for the weaker performance of stratification was the reduced effectiveness of batch sampling with this partitioning technique, see Sec. 3.5.6. Increasing the number of negative samples by 10× mildened this effect and led to an overall increased embedding quality (see also Tab. 3.10). Graph-cut partitioning further reduced communication cost by 5-6×, but did not provide faster epoch times due to unbalanced triple partition sizes.

	Set up	Partitioning technique	Epoch time	Time to 0.95 MRR	MRR	Hits @10
FB15K	1@1	Seq. (GPU memory)	9.5s	11.9min	0.705	0.834
	1@1	Seq. (main memory)	11.4s	14.3min	0.705	0.834
	2@1	Stratification (CARL)	**4.6s**	**5.8min**	**0.725**	**0.835**
	1@2	Stratification (CARL)	5.9s	7.4min	**0.725**	**0.835**
Yago3-10	1@1	Seq. (GPU memory)	74.1s	259.3min	**0.451**	**0.637**
	1@1	Seq. (main memory)	88.0s	307.8min	**0.451**	**0.637**
	2@1	Stratification (CARL)	**40.8s**	**166.6min**	0.438	0.607
	1@2	Stratification (CARL)	43.3s	175.8min	0.438	0.607
Wikidata5M	1@1	Seq. (GPU memory)	798.4s	199.6min	0.258	**0.348**
	1@1	Seq. (main memory)	985.7s	246.4min	0.258	**0.348**
	2@1	Stratification (ARL)	**466.7s**	**77.8min**	**0.264**	0.344
	1@2	Stratification (ARL)	477.7s	79.6min	**0.264**	0.344
Freebase	1@1	Seq. (main memory)	6495.7s	-	0.566	0.627
	4@1	Random (RLB)	1290.8s	-	**0.567**	**0.630**
	2@2	Random (RLB)	1541.4s	-	**0.567**	**0.630**
	4@2	Random (RLB)	**938.3s**	-	0.562	0.621

Table 3.8: Partitioning techniques (best-performing variant in terms of time to 0.95 MRR). Calculating time to 0.95 MRR was too expensive for Freebase. (RotatE)

	Rand. (R)	Rand. (RL)	Rel.	GC (L)	Strat. (CARL)
Total time	338±6	275±0	315±1	232±54	227±0
Proc. time	228±4	218±0	219±4	193±44	217±2
Wait time	110±1	63±0	96±3	39.3±10	11±2

Table 3.9: Avg. processing and wait time in seconds per worker and epoch with std. dev. (ComplEx, 1@2, Wikidata5M).

Tab. 3.9 shows processing and wait time per worker and epoch for each partitioning technique. Stratification (CARL) and graph-cut heavily reduced wait times and therefore led to the shortest overall processing times. However, graph-cut showed a high variance between workers; see Sec. 3.5.7 for additional analysis on graph-cut.

3.5.4 Comparison to Original Work (Tab. 3.11)

The above results were all obtained using an independent implementation of parallel training. It is challenging to put these results in perspective with the results originally reported by the DGL-KE and PBG frameworks. The reason is that these prior studies (i) did not report MRR but *sampled MRR* (sMRR) and (ii) used different variants of sMRR. In general, sMRR/X is obtained by ranking test triples against a set of X

	Set-up	Partition technique	Epoch time	Data sent per epoch	sMRR /1,000	MRR	Hits @10
	1@1	Seq. (mm)	3929s	-	0.811	0.364	0.487
	1@1	Seq. (B) (mm)	3925s	-	0.815	0.426	0.528
	2@2	Random (RLB)	966s	232.8GB	**0.816**	**0.426**	**0.529**
	2@2	Relation (rLB)	823s	205.9GB	0.801	0.397	0.507
	2@2	Strat. (CARLB)	**803s**	123.2GB	0.793	0.325	0.424
	2@2	Graph-cut (LB)	1170s	**42.5GB**	0.789	0.407	0.512
	4@2	Random (RLB)	592s	251.9GB	0.819	0.421	0.523
10× neg.	2@2	Random (RLB)	1481s	232.8GB	**0.841**	**0.478**	**0.588**
10× neg.	2@2	Relation (rLB)	1341s	205.9GB	0.808	0.454	0.569
10× neg.	2@2	Strat. (CARLB)	**1127s**	122.1GB	0.798	0.451	0.556
10× neg.	2@2	Graph-cut (LB)	1810s	**44.4GB**	0.786	0.467	0.567

Table 3.10: Comparison of partitioning techniques (ComplEx, Freebase). Note that the sampled MRR (sMRR) metrics used in some prior studies are misleading. The lower part shows results with 10× # negatives.

Setup	Frame-work	Partition technique	Dim.	Epoch Time	MRR	Hits@10
2@2	PBG	Strat. (LB)	100	2856s	0.157	0.250
2@2	Ours	Strat. (LB)	100	1983s	0.291	0.384
4@1	DGL-KE	Relation (B)	400	~600s[j]	0.614	0.665
4@1	Ours	Relation (B)	400	860s	0.612	0.662

Table 3.11: Performance comparison to original implementations (ComplEx, Freebase).

random triples instead of all triples. Note that sMRR/X decreases with increasing X until it reaches the MRR. The use of sMRR is computationally cheaper, but we found that it produced misleading results. As can be seen in Tab. 3.10, sMRR highly overestimated MRR, and, perhaps more importantly, different models suggested similar quality in terms of sMRR even when their MRR differed substantially. One reason for this misleading approximation is that "hard negative" entities (which appear before the target entity in the full ranking) are unlikely to be sampled. Addressing this issue (Lerer et al. 2019) and (Zheng et al. 2020) proposed a batch sampling approach. We did not consider these methods because they did not solve the problematic approximation in preliminary experiments and resulted in an evaluation metric that is batch size dependent. For this reason, we use MRR, even though the computation is expensive.

[j]This epoch time is approximated since DGL-KE does not have a concept of epochs but only reports in steps.

	Setting	N_s^-	N_o^-	Epoch time	Data sent per epoch	MRR	Hits @10
Yago3-10	Uniform	892	894	114.9s	66.5GB	0.518	0.659
	Shared	892	894	9.3s	1.9GB	0.508	0.649
	Shared	8919	8942	21.1s	7.2GB	0.538	0.672
Wiki-data5M	Uniform	66	236	5112.3s	3262.0GB	0.218	0.325
	Shared	66	236	153.6s	24.6GB	0.204	0.310
	Shared	2176	7851	347.2s	114.2GB	0.296	0.395

Table 3.12: Shared sampling reduced communication overhead and epoch time (ComplEx, 1@2, Random (R)).

To get some intuition into the framework performance, we trained a ComplEx model with DGL-KE and PBG with their provided hyperparameter settings. We then trained a corresponding model in our framework (using the same dimensionality). The results in Tab. 3.11 suggest that our implementation is competitive.

3.5.5 Sampling Techniques (Tab. 3.12- 3.14)

Shared Sampling. To investigate the effect of shared sampling, we ran a hyperparameter search for the model ComplEx with uniform sampling (1@1) to obtain a suitable choice of the numbers N_s^- and N_o^- of negative samples. We fixed this configuration but (i) switched to shared sampling to measure the impact on quality and (ii) used a 1@2 setup with random partitioning to measure the impact on communication cost. Our results are summarized in Tab. 3.12.

First, observe that shared sampling is very efficient: its use led to a 40x reduction in network footprint and a 7x reduction in epoch time. However, the model quality suffered in that the resulting MRR decreased. This is a consequence of the reduced variety of negative samples introduced by shared sampling. We found that the quality degradation could be countered by using a large number of negative samples. We reran the hyperparameter search with shared sampling and a 10x larger limit in the upper bound on the number of samples; the results are also reported in Tab. 3.12. As can be seen, the increased number of samples led to models of similar or better quality than those obtained with uniform sampling but was substantially more efficient in terms of epoch time and network footprint. The lower part of Tab. 3.10 shows that these quality improvements also hold for the largest dataset Freebase.

Local Sampling. To study the impact of local sampling, we first investigated the performance obtained by the various partitioning techniques with shared sampling with and without local sampling; see Tab. 3.13. We found that none of the partitioning

	Sample from:	All Entities			Local Entities		
		Time	Comm.	MRR	Time	Comm.	MRR
FB15K	Random (R)	2.9s	0.7GB	**0.775**	2.9s	0.5GB	**0.775**
	Relation	**2.8s**	0.6GB	0.771	**2.8s**	0.4GB	0.729
	Strat. (CAR)	3.5s	0.3GB	0.771	2.9s	**0.1GB**	0.765
	Graph-cut	3.3s	0.3GB	0.766	3.3s	**0.1GB**	0.506
Yago3-10	Random (R)	21.1s	7.2GB	0.538	19.5s	1.3GB	**0.547**
	Relation	23.6s	7.1GB	0.538	22.6s	1.3GB	0.497
	Strat. (R)	29.8s	12.1GB	0.534	**13.8s**	1.2GB	0.531
	Graph-cut	28.3s	11.2GB	0.535	18.2s	**0.2GB**	0.211
Wikidata5M	Random (R)	347.2s	114.2GB	0.296	290.6s	27.2GB	0.297
	Relation	320.5s	111.4GB	0.296	275.6s	24.1GB	0.290
	Strat. (CAR)	393.1s	143.7GB	0.291	**228.0s**	15.0GB	**0.308**
	Graph-cut	417.4s	137.7GB	0.294	317.2s	**6.1GB**	0.192

Table 3.13: Local sampling reduced epoch time. The static sampling pool of relation/graph-cut can have a negative influence on quality. Best per dataset marked in bold (ComplEx, 1@2).

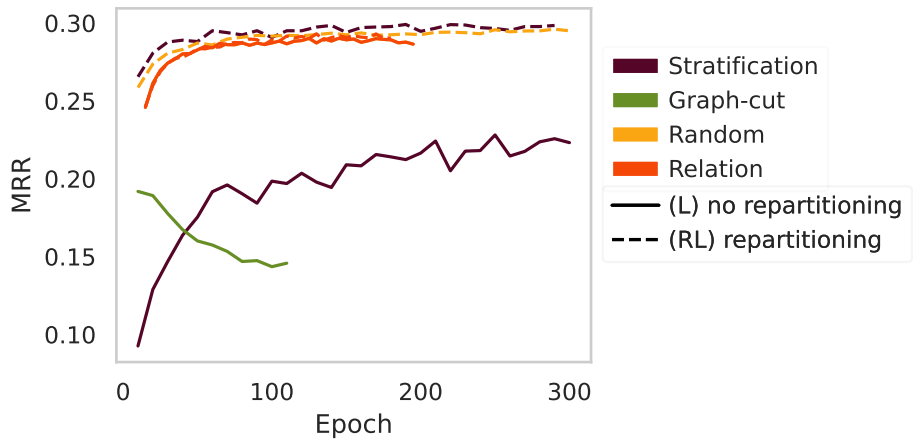


Figure 3.5: Local sampling without repartitioning led to substantial drops in embedding quality for stratification and graph-cut. (ComplEx, Wikidata5M)

techniques provided substantial benefits w.r.t. random partitioning without the use of local sampling. When local sampling was used, epoch times and communication cost decreased for all partitioning techniques on all datasets; stratification benefited the most.

While local sampling may be efficient, it is not always effective as it may decrease quality. To gain insight into why this is the case, Fig. 3.5 shows the progress of validation MRR during training for various techniques. Note that there was a substantial drop in validation MRR for stratification and graph-cut partitioning once

Dataset	Batch Neg.	Time	Comm.	MRR	Hits@10
FB15K	0%	2.9s	0.7GB	**0.775**	**0.859**
	50%	2.9s	**0.6GB**	0.766	0.852
	100%	**2.7s**	**0.6GB**	0.745	0.845
Yago3-10	0%	21.1s	7.2GB	**0.538**	**0.672**
	50%	14.4s	5.6GB	0.422	0.589
	100%	**9.1s**	**1.0GB**	0.375	0.555
Wiki-data5M	0%	347.2s	114.2GB	**0.296**	**0.395**
	50%	230.8s	72.3GB	0.224	0.321
	100%	**142.6s**	**19.4GB**	0.186	0.274
Free-base	0%	983.8s	295.0GB	0.364	0.483
	50%	**944.7s**	263.4GB	**0.420**	**0.525**
	100%	948.2s	**232.0GB**	0.405	0.512

Table 3.14: Batch sampling was dataset-dependent. Best per dataset marked in bold (ComplEx, 1@2, Random (R)).

local sampling was used. Stratification with repartitioning (as also done in CARL stratification) remained unaffected. Since repartitioning changes the sampling pool from epoch to epoch, the quality degradation was avoided.

We conclude that local sampling was generally efficient, but required data repartitioning between epochs to avoid large drops in embedding quality.

Batch Sampling. We analyzed the effect of batch sampling next. We trained ComplEx in a 2@1 setup. For each setting, we trained with 0%, 50%, and 100% of entities sampled from within the batch (and the rest shared). The results are shown in Tab. 3.14. We found that the substantial reduction of the sampling pool performed by batch sampling consistently deteriorated embedding quality on all datasets but Freebase, where batch sampling substantially improved embedding quality. Results are thus highly dataset-dependent. Over all datasets a combination of uniform and batch sampling was preferable. To see whether the quality degradation can be avoided, we performed a separate hyperparameter search with batch sampling enabled but were not able to reach a similar quality as achieved without batch sampling.

3.5.6 Combination of Sampling and Partitioning Techniques (Tab. 3.15)

We investigated the interplay of partitioning techniques with local sampling as well as batch sampling on Freebase, the only dataset for which we saw a positive influence of batch sampling. Our results for random and stratification partitioning are shown in

Set-up	Partition technique	Epoch time	Data sent per epoch	MRR	Hits @10
1@1	Seq. (mm)	3929s	-	0.364	0.487
1@1	Seq. (B) (mm)	3925s	-	**0.426**	0.528
2@2	Random (R)	983s	295.0GB	0.364	0.483
2@2	Random (RL)	953s	232.0GB	0.365	0.485
2@2	Random (RB)	944s	263.4GB	0.420	0.525
2@2	Random (RLB)	966s	232.8GB	**0.426**	**0.529**
2@2	Strat. (CARL)	819s	124.3GB	0.327	0.439
2@2	Strat. (CARB)	1128s	182.4GB	0.391	0.491
2@2	Strat. (CARLB)	**803s**	**123.2GB**	0.325	0.424

Table 3.15: Comparison of sampling technique combinations (ComplEx, Freebase).

Dataset	#Partitions	#Triples			Epoch time	
		CV	Min	Max	Min	Max
FB15K	8	24.9%	6.3%	17.4%	0.8s	2.0s
Yago3-10	8	37.1%	7.3%	22.0%	5.6s	14.8s
Wikidata5M	8	28.9%	7.3%	17.3%	92.4s	194.7s
Freebase	8	15.9%	9.4%	15.1%	587.9s	1170.6s

Table 3.16: Graph-cut leads to unbalanced triple partitions. CV is the coefficient of variation of the triple partition sizes.

	FB15K	Yago3-10	Wikidata5M	Freebase
Normal	78.6%	37.3%	27.3%	25.9%
Combined	82.6%	62.0%	43.2%	31.3%

Table 3.17: Stratification - average fraction of active entities per partition ($M = 8$).

Tab. 3.15. Batch sampling was generally beneficial, whereas a combination of local and batch sampling deteriorated quality with stratification but not random partitioning.

3.5.7 Graph-Cut Partitioning (Tab. 3.16)

In our experiments, we often found that graph-cut partitioning led to lower speedups than CARL stratification (in addition to often leading to deteriorating embedding quality). The reason was that triple partition sizes were often quite unbalanced. Tab. 3.16 shows that this holds for all datasets and that the time required to process a partition varied substantially between the smallest and largest partition.

	Dataset	All entities		Active entities	
		Epoch time	Data sent	Epoch time	Data sent
Normal	FB15K	3.4s	0.2GB	3.6s	0.2GB
	Yago3-10	13.8s	1.2GB	10.7s	0.4GB
	Wikidata5M	296.7s	60.3GB	245.6s	26.5GB
Com-bined	FB15K	3.0s	0.1GB	2.9s	0.1GB
	Yago3-10	11.3s	0.6GB	10.9s	0.4GB
	Wikidata5M	252.3s	31.0GB	228.0s	15.0GB

Table 3.18: With stratification, sampling only from active entities (A), and combining mirror partitions (C) network footprint and epoch time decreased (ComplEx 1@2).

3.5.8 Plain Stratification (Tab. 3.18)

We used the improved CARL stratification throughout the experimental study because it was consistently more efficient and effective than plain stratification. Here we report on the influence of combining mirror partitions (C), relocating only active entities (A), and repartitioning (R). We trained various variants of stratification in a 2@1 setting and measured training time, embedding quality, and communication costs.

Tab. 3.17 lists the average fraction of active entities for stratification with $M = 8$ with and without combining mirror partitions. Note that for the larger datasets, most entities are inactive, i.e., they do not occur in the respective partition. As shown in Tab. 3.18, the restriction of embedding relocation to active entities substantially reduced the network footprint (up to 70%) and also improved epoch times (up to almost 20%). Combining partitions further reduced the network footprint and communication cost as embeddings needed to be synchronized less often.

Repartitioning (R) mainly affected quality; see Fig. 3.6. In fact, stratification without repartitioning led to a substantial drop in quality due to local sampling, as discussed in Sec. 3.5.5. Stratification with repartitioning as well as CARL stratification did not lead to reduced quality. In contrast, we observed a small improvement with CARL stratification. This may be due to the change of sampling bias when using active entities: it is closer to the actual distribution of entities in the KG.

Overall, CARL stratification was instrumental for the effectiveness of stratification; plain stratification was not competitive.

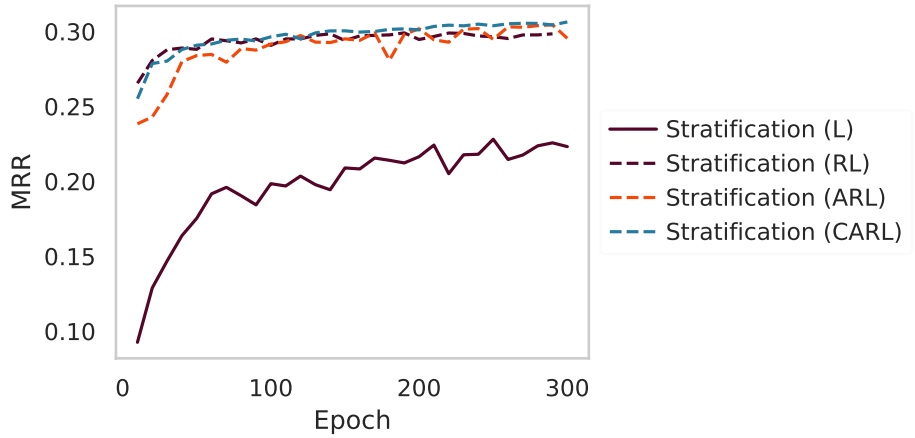


Figure 3.6: Repartitioning (R) and sampling only from active entities (A) had a positive influence on quality (ComplEx, Wikidata5M).

Partition Technique	Data sent		MRR	
	Adagrad	Row-Adagrad	Adagrad	Row-Adagrad
Sequential	-	-	0.297	0.291
Random (R)	125.2GB	65.7GB	0.296	0.298
Relation	123.8GB	63.5GB	0.296	0.300
Stratification (CARL)	15.0GB	7.4GB	0.308	0.306
Graph-cut	6.1GB	3.7GB	0.192	0.181

Table 3.19: Row-wise optimizers reduced storage overhead and communication cost (ComplEx, 1@2, Wikidata5M)

3.5.9 Row-Wise Optimizers (Tab. 3.19)

Row-wise optimizers treat each embedding as a single parameter instead of each dimension of an embedding and therefore reduce storage and communication overhead by about 50%. We observed no negative influence on the resulting embedding quality for all partitioning methods but graph-cut partitioning, where the drop was small but noticeable. Overall, we found Row-Adagrad to be a suitable approach to reduce storage and communication costs; see Tab. 3.19

3.5.10 Influence of Hyperparameter Search (Tab. 3.20)

As hyperparameter search for KGEs on large-scale graphs is expensive, this study followed previous work (Lerer et al. 2019; Zheng et al. 2020) and tuned models for Freebase on the smaller benchmark FB15K based on the same KG. However, the resulting quality is suboptimal. Therefore, we analyzed whether the choice of hyperparameters had an impact on takeaways on parallel training techniques. For

Partition Technique	HPO on FB15K		HPO with GraSH	
	no (B)	(B)	no (B)	(B)
Random (RL)	0.365	**0.426**	0.622	**0.642**
Relation (rL)	0.306	0.397	0.463	0.621
GraphCut (L)	0.317	0.407	0.485	0.629
Stratification (CARL)	0.328	0.325	0.589	0.591

(a) Influence of partitioning technique did not change.

Batch Neg.	HPO on FB15K	HPO with GraSH
0%	0.364	0.594
50%	**0.420**	**0.642**
100%	0.405	0.609

(b) Influence of batch sampling did not change (Random (R)).

Table 3.20: Influence on MRR of parallel training techniques did not change with improved hyperparameters. Best training setting per hyperparameter setting marked in bold (ComplEx, 4@1, Freebase).

efficient and effective hyperparameter optimization, we used GraSH. For more details on GraSH and large-scale HPO see Ch. 4. Results are summarized in Tab. 3.20.

Mainly, the influence of parallel training techniques stayed the same; findings were stable over hyperparameters. The best training approach stayed random partitioning with repartitioning and a combination of local and batch sampling. Further, we saw a similar strong influence of batch sampling on all partitioning techniques except stratification. But the performance of stratification is stronger and closer to other partitioning techniques with proper hyperparameter tuning.

3.6 Parallel & Subsequent Work

3.6.1 Strata Scheduling

Marius (Mohoney et al. 2021), developed in parallel to this study, focuses on efficient training of (knowledge) graph embeddings on a single machine with a single GPU. It relies on stratification partitioning with two main improvements; (i) improved pipelining, and (ii) improved partition scheduling. For (i), Marius allows for stale embeddings. In particular, it loads the embeddings needed for the upcoming batch in parallel to writing updates of the current batch. W.r.t. (ii), Marius ensures with the so-called Beta ordering that only one of the two entity partitions forming a triple partition in stratification is exchanged after processing. As the underlying stratification partitioning does not use repartitioning, we expect a negative impact of this approach as seen in Sec. 3.5.8.

Marius-GNN. Next to improvements for large-scale GNN training, the extension Marius-GNN (Waleffe et al. 2023) further improves on the Beta scheduling and offers

a dynamic approach of stratification termed COMET. COMET builds smaller entity partitions compared to Beta and allows for more randomness during scheduling. Here, the schedule is changed every epoch. This approach is closely related to our repartitioning approach presented in Sec. 3.3.6.

GE2 (Zheng et al. 2024) offers an additional improvement to stratification scheduling. While the Beta and COMET scheduling already reduced communication overhead, these scheduling approaches were built for single GPU training only. GE2 addresses this shortcoming and further reduces communication overhead for multi-GPU training with stratification partitioning. It builds a schedule by framing the arrangement of partitions as the RBIBD problem (Kirkman 1847). To ensure high quality it utilizes repartitioning as proposed in Sec. 3.3.6.

3.6.2 Parameter Management

In this work, we used the parameter server Lapse (Renz-Wieland et al. 2020) to relocate embeddings and manage (concurrent) embedding access. The follow-up work AdaPM (Renz-Wieland et al. 2023) additionally allows (i) to signal intents when embeddings are needed in later training steps and (ii) for a mixture between embedding relocation and replication. These two additions allow for more efficient training, especially with random partitioning. In this setting, embeddings of hot spot entities like high-degree entities can be automatically replicated when necessary, and sparsely accessed embeddings relocated before they are needed. This approach can further hide latencies occurring especially in the multi-machine setup. However, AdaPM was evaluated for CPU training only, and does not offer direct relocation to GPU. Further integrating GPUs in the parameter management process, fully sharded data parallel (FSDP) (Zhao et al. 2023), integrated in PyTorch, allows to shard embeddings over VRAM of multiple GPUs and CPU RAM, as well as for asynchronous prefetching of needed embeddings and/or activations. Next to the training of dense Transformer models, this approach showed promising results in training recommendation models with sparse parameter accesses similar to KGE training. Making the parameter server “GPU-native“ and relocating or replicating embeddings directly to the device would reduce latency. However, the impact of this latency reduction would differ per partitioning and negative sampling technique. Therefore, future studies should explore if the findings of this study hold if such latency-hiding techniques were applied.

3.7 Conclusion

We described and evaluated state-of-the-art techniques for parallel training of KGE models of large-scale knowledge graphs. We found that it is possible to achieve high speedup (up to 7× with 8 GPUs) with high embedding quality, both in a single-machine multi-GPU and in a multi-machine multi-GPU setup. However, the parallelization techniques currently implemented in large-scale KGE training frameworks did not realize these improvements and often led to quality degradation compared to sequential training. This was mainly caused by the combination of static partitioning and local sampling used by these implementations. Our experiments suggest that the overall choice of partitioning and sampling technique is highly dataset-dependent. For example, on Freebase, the random partitioning baseline in combination with improved sampling methods led to the overall best results. On all other datasets, CARL stratification—a variant of stratification as used in PyTorch BigGraph—along with shared local sampling often performed competitive or best.

In subsequent studies, several enhancements to existing parallelization frameworks have been proposed (see Sec. 3.6). Despite these advancements, we anticipate that the findings presented in this study remain valid when integrated with these new improvements. Notably, recent developments in parameter management have significantly enhanced latency hiding and reduced the communication costs of parameter servers. These advancements, coupled with the strong performance of the random partitioning baseline demonstrated in this study, suggest that contemporary parallel training of KGEs should leverage random partitioning in conjunction with improved parameter servers.

3.7.1 Limitations

With this study, we implemented and improved techniques presented in previous works in a common framework. Even though the underlying Python implementation mainly relies on C-based libraries, it puts a limit on the execution time of each parallel training technique. For more efficient training, the most promising techniques should be implemented, e.g., in C/C++, avoiding Pythons thread-lock and allowing for improved pipelining approaches.

Further, this study was performed using GPUs with a VRAM of 11 GB. A larger VRAM size could impact the communication overhead between workers as well as the embedding relocation overhead between RAM and VRAM. E.g., sets of embeddings of larger stratification partitions could be allocated directly on GPU, reducing cost of partition swaps.

Additionally, we only studied graph and model sizes for which all embeddings fit into main memory. While the decreasing cost of main memory allows scaling to larger model sizes, all embeddings for larger graphs as present in industry might not fit into main memory. The effects of loading embeddings from disk on training speed and the impact of pipelining approaches on model quality should be analyzed in future work.

While we evaluated the effect of large-scale HPO on parallel training techniques (Sec. 3.5.10), we did not perform a separate hyperparameter search per partitioning and sampling technique. A proper HPO per technique could further influence the resulting model quality.

Finally, improved strata scheduling techniques (Sec. 3.6.1) introduced after this study may influence performance of stratification partitioning, and later introduced parameter management techniques (Sec. 3.6.2) hide and reduce the latency of embedding exchanges. While we assume that improved parameter management techniques have an especially positive impact on simple techniques like random partitioning, it should be investigated, to what extent these newly introduced approaches influence the findings of this study.

Chapter 4

Hyperparameter Tuning for Large-Scale Knowledge Graph Embedding Models

"Harpists spend 90% of their lives tuning their harps and 10% playing out of tune."

Igor Stravinsky, n.d.

In the previous chapter, we have shown that efficient and effective training of large-scale KGE models is possible, but the resulting quality depends on underlying training techniques. In addition to training techniques, KGE models are sensitive to hyperparameter settings, with suitable choices being model and dataset-dependent (Ali et al. 2021a; Ruffinelli et al. 2020). However, for large KGs (such as Freebase), the cost of evaluating individual hyperparameter configurations is excessive. In the previous chapter, we as well as prior studies (Lerer et al. 2019; Zheng et al. 2020) avoided this cost by using various heuristics, e.g., by training on a subgraph or by using fewer epochs. In this chapter, we systematically discuss and evaluate the quality and cost savings of such heuristics and other low-cost approximation techniques. Based on our findings, we introduce GraSH, an efficient multi-fidelity hyperparameter optimization (HPO) algorithm for large-scale KGEs that combines both graph and epoch reduction techniques and runs in multiple rounds of increasing fidelities.

This chapter is based on Kochsiek et al. (2022).

4.1 Introduction

Prior studies have shown that embedding quality is highly sensitive to the hyperparameter choices used when training the KGE model (Ali et al. 2021a; Ruffinelli et al. 2020). Moreover, the search space is large and hyperparameter choices are dataset- and model-dependent. For example, the best configuration found for one model may perform badly for a different model. As a consequence, we generally cannot transfer suitable hyperparameter configurations from one dataset to another or from one KGE model to another. Instead, a separate hyperparameter search is often necessary to achieve high-quality embeddings. However, the positive impact of a well-performed hyperparameter search is drastic; e.g., a performance increase of $\approx$ 20 percentage points in MRR is possible over the use of heuristics as done in Ch. 3.

Although using an extensive hyperparameter search may be feasible for smaller datasets—e.g., the study of Ruffinelli et al. (2020) uses 200 configurations per dataset and model—, such an approach is generally not cost-efficient or even infeasible on large-scale KGs, where KGE training is expensive in terms of runtime, memory consumption, and storage cost. For example, the Freebase KG consists of $\approx$ 86 M entities and more than 300 M triples. A single training run of a 512-dimensional ComplEx embedding model on Freebase takes up to 50 min per epoch utilizing 4 GPUs and requires $\approx$ 164 GB of memory to store the model. Therefore, following Ruffinelli et al. (2020) and tuning a single model in such a setting would take more than two months.

To reduce these excessive costs, prior studies on large-scale KGE models either avoid hyperparameter optimization (HPO) altogether or reduce runtime and memory consumption by employing various heuristics. The former approach leads to suboptimal quality, whereas the impact in terms of quality and cost of the heuristics used in the latter approach has not been studied in a principled way. The perhaps simplest of such heuristics is to evaluate a given hyperparameter configuration using only a small number of training epochs (e.g., in Ch. 3 we only used 20 epochs for HPO on the Wikidata5M dataset). Another approach is to use a small subset of the large KG (e.g., the small FB15K benchmark dataset instead of full Freebase) to obtain a suitable hyperparameter configuration (Lerer et al. 2019; Zheng et al. 2020) or a set of candidate configurations (Zhang et al. 2022). The general idea behind these heuristics is to employ *low-fidelity approximations* (fewer epochs, smaller graph) to compare the performance of different hyperparameter configurations during HPO, before training the final model on *full fidelity* (many epochs, entire graph).

In this chapter, we explore how to effectively use a given HPO budget to obtain a high-quality KGE model. To do so, we first summarize and analyze both cost and quality of various low-fidelity approximation techniques. We found that there are substantial differences between techniques and that a combination of reducing the number of training epochs and the graph size is generally preferable. To reduce KG size, we propose to use its *k-core* subgraphs (Seidman 1983); this simple approach worked best throughout our study.

Building upon these results, we present GraSH (short for **gra**ph **s**uccessive **h**alving), an efficient HPO algorithm for large-scale KGE models. At its heart, GraSH is based on successive halving (SH) (Jamieson and Talwalkar 2016). It uses multiple fidelities and employs several KGE-specific techniques, most notably, a simple cost model, negative sample scaling, subgraph validation, and a careful choice of fidelities. We conducted an extensive experimental study and found that GraSH achieved state-of-the-art results on large-scale KGs with a low overall search budget corresponding to only three complete training runs. For example, using GraSH, the model ComplEx performed up to 20 percentage points better on the full Freebase KG, in terms of MRR, compared to a similar setting using heuristics (c.f. results in Ch. 3 and Zheng et al. (2024)). To reach such high quality with low resource consumption, both the use of multiple reduction techniques simultaneously and of multiple fidelity levels was key.

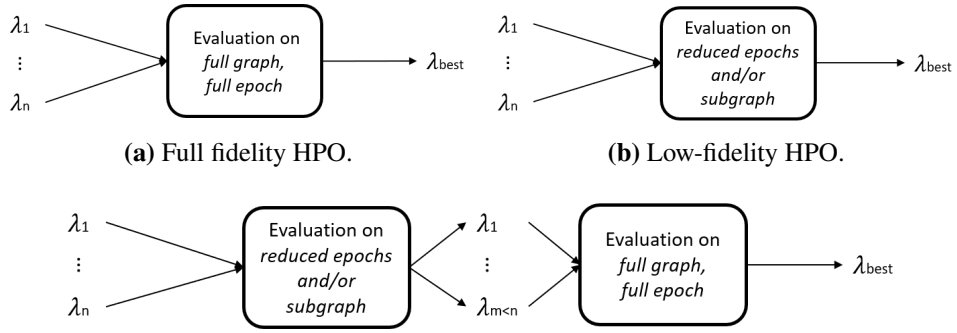
4.2 Related Work

GraSH is a multi-fidelity HPO algorithm for KGEs and builds upon findings of KGE-specific tuning approaches as well as multi-fidelity search algorithms. In the following, we present relevant approaches of both categories.

4.2.1 HPO for KGEs

HPO algorithms for KGEs can be categorized into three groups: full-fidelity, low-fidelity, and two-stage HPO. The three categories are summarized in Fig. 4.1.

Full-fidelity HPO (Fig. 4.1a). Recent studies analyzed the impact of hyperparameters and training techniques for KGE models using full-fidelity HPO (Ali et al. 2021a; Ruffinelli et al. 2020). In these studies, the vast hyperparameter search space was explored using a random search and Bayesian optimization with more than 200 full training runs per model and dataset. The studies focused on smaller benchmark KGs, however; such an approach is excessive for large-scale knowledge graphs.



(a) Full fidelity HPO. **(b)** Low-fidelity HPO.

(c) Two-stage HPO.

Figure 4.1: Schematic illustration of HPO approaches for KGEs. λ describes a single hyperparameter configuration.

Low-fidelity HPO (Fig. 4.1b). Recent work on large-scale KGE models circumvented the high cost of full-fidelity HPO by relying on low-fidelity approximations such as epoch reduction (Ch. 3) and using smaller benchmark graphs (Ch. 3 & (Lerer et al. 2019; Zheng et al. 2020)) in a heuristic fashion. The best-performing hyperparameters in low-cost approximations were directly applied to train a single full-fidelity model. Our experimental study suggests that such an approach may neither be cost-efficient nor produce high-quality results.

Two-stage HPO (Fig. 4.1c). AutoNE (Tu et al. 2019) is an HPO approach for training large-scale network embeddings that optimizes hyperparameters in two stages. It first approximates hyperparameter performance on subgraphs created by random walks, a technique we will explore in Sec. 4.4. Subsequently, AutoNE transfers these results to the full graph using a meta-learner. In the context of KGs, this approach was outperformed by KGTuner (Zhang et al. 2022),[a] which uses a multi-start random walk (fixed to 20% of the entities) in the first stage and evaluates the top-performing configurations (fixed to 10) at full fidelity in the second stage. Such fixed heuristics often limit flexibility in terms of budget allocation and lead to an expensive second stage on large KGs. In contrast, GRASH makes use of multiple fidelity levels, carefully constructs and evaluates low-fidelity approximations, and adheres to a prespecified overall search budget. These properties are key for large KGs; see Sec. 4.5.3 for an experimental comparison with KGTuner.

4.2.2 Multi-Fidelity HPO

Successive halving (SH) (Jamieson and Talwalkar 2016) is a general HPO approach that selects hyperparameters over multiple rounds of increasing fidelity. Typically,

[a] KGTuner was proposed in parallel to this work.

the fidelity is either defined by the number of training epochs or training examples. The approach works as follows:

(i) *Initialization.* Define/sample an initial set of hyperparameter configurations and overall budget for HPO.

(ii) *Evaluation.* (Train and) evaluate model quality on fidelity based on round budget. The round budget is dependent on the overall HPO budget and reduction factor η.

(iii) *Halving.* Select and keep the fraction of $\lceil 1/\eta \rceil$ best-performing hyperparameter configuration. By default $\eta = 2$.

(iv) *Succession.* Increase fidelity, repeat steps (ii) and (iii) until only a single configuration is left.

For an exemplified visualization of GraSH, an extension of SH, see Fig. 4.2. Note that the runtime of SH can be improved by asynchronous execution (Li et al. 2020), i.e., increasing fidelity before step (ii) finished.

Hyperband (Li et al. 2017) is an extension of successive halving and uses it as a subroutine. Successive halving requires the number of initial configurations as well as the overall search budget as input. The choice of whether to evaluate more configurations on a lower fidelity or to evaluate a smaller number of configurations on a higher fidelity can have a significant impact on resulting quality. Therefore, these parameters can be seen as hyperhyperparameters. Hyperband addresses this tradeoff by running successive halving multiple times with varying combinations of number of trials and training budget.

BOHB (Falkner et al. 2018) further extends hyperband by integrating Bayesian optimization. In particular, it uses hyperband to determine how many configurations to evaluate per fidelity but replaces the random selection of configurations with a model-based search. As soon as enough data points are gathered (i.e., configurations evaluated), BOHB fits a tree parzen estimator (Bergstra et al. 2011) surrogate model per fidelity. It always uses the surrogate model fit using the highest fidelity, for which enough observations are available. For robustness, this approach is combined with continued random sampling of configurations.

MFES-HB (Li et al. 2021)**.** Hyperband discards most configuration evaluations early, ending up with many low-fidelity measurements and only a few high-fidelity ones. The Bayesian optimization integrated in BOHB only utilizes the few high-fidelity optimizations. Addressing this limitation with MFES-HB, Li et al. (2021) train an ensemble surrogate model over multiple fidelity levels.

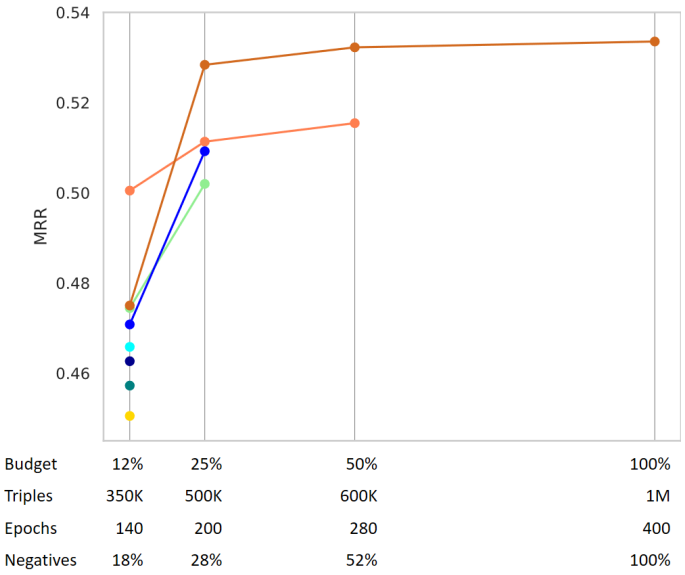


Figure 4.2: Example run of GraSH starting with 8 configurations on Yago3-10, using a combined reduction technique of epoch and graph reduction, and reducing with factor $\eta = 2$.

While the techniques presented in this study rely on successive halving, the underlying multi-fidelity search algorithm can be exchanged by the here presented approaches for possible further improvements.

4.3 Successive Halving for Knowledge Graphs (GraSH)

GraSH is a multi-fidelity HPO algorithm for KGE models based on successive halving (Jamieson and Talwalkar 2016). As successive halving, GraSH proceeds in multiple *rounds* of increasing fidelity; only the best configurations from each round are transferred to the next round. An example run of GraSH is visualized in Fig. 4.2. In contrast to other KGE-specific HPO techniques discussed in Sec. 4.2.1, this approach allows to discard of unpromising configurations at very low cost. GraSH differs from successive halving mainly in its parameterization and its use of KG-specific reduction and validation techniques.

Parameterization. GraSH is summarized in Alg. 4. Given knowledge graph $\mathcal{G}$, GraSH outputs a single optimized hyperparameter configuration. GraSH is parameterized as described in Alg. 4; default parameter values are given if applicable. The most important parameters are the maximal number E of epochs and the overall search budget B. The search budget B is relative to the cost of a full training run, which in turn is determined by E. The default choice $B = 3$, for example, corresponds to an overall search cost of three full training runs. We chose this parameterization

Algorithm 4 GraSH: Successive halving for knowledge graph embeddings

Require:
KG $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{K})$,
max. epochs E,
search budget B, default: 3,
num. configurations n, default: 64,
reduction factor η, default: 4,
reduction method $r \in \{\textit{epoch}, \textit{graph}, \textit{combined}\}$, default: *combined*
Ensure: Hyperparameter configuration

$s \leftarrow \lceil \log_\eta(n) \rceil$ ▷ Number of rounds
$R \leftarrow B/s$ ▷ Per-round budget
$\Lambda_1 \leftarrow$ generate configurations $\{\lambda_1, \ldots, \lambda_n\}$ ▷ n hyperparameter configurations
for $i \in \{1, \ldots, s\}$ **do** ▷ i-th round
  $f_i \leftarrow R/|\Lambda_i|$ **if** $v \neq$ *combined* **else** $R/\sqrt{|\Lambda_i|}$ ▷ Target fidelity
  $E_i \leftarrow f_i E$ **if** $r \neq$ *graph* **else** E ▷ Epochs in round i
  $\mathcal{G}_i \leftarrow$ reduced KG with $f_i|\mathcal{K}|$ triples **if** $v \neq$ *epoch* **else** $\mathcal{G}$ ▷ Graph in round i
  $\mathcal{G}_i^{\text{train}}, \mathcal{G}_i^{\text{valid}} \leftarrow$ create train and valid split of $\mathcal{G}_i$
  $V_i \leftarrow \emptyset$
  for $\lambda \in \Lambda_i$ **do**
    $\lambda \leftarrow$ adapt λ to $\mathcal{G}_i^{\text{train}}$ by scaling negatives
    $m \leftarrow$ train a model with λ on $\mathcal{G}_i^{\text{train}}$ for E_i epochs
    $V_i \leftarrow V_i \cup \{\text{validate } m \text{ on } \mathcal{G}_i^{\text{valid}}\}$
  end for
  $\Lambda_{i+1} \leftarrow$ best $\lceil |\Lambda_i|/\eta \rceil$ configurations from Λ_i according to V_i
end for
return Λ_{s+1} ▷ Only single configuration left

because it is independent of utilized hardware and both intuitive and well-controllable. The reduction factor η controls the number of configurations (starts at n, decreases by factor of η per round) and fidelity (increases by factor of η) of each round. Note that GraSH does not train at full fidelity, i.e., its final configuration still needs to be trained on the full KG (not part of budget B). Finally, GraSH is parameterized by a *reduction method* r. This parameter controls which reduction technique to use (only epoch, only graph, or combined).

Algorithm overview. Like successive halving, GraSH proceeds in rounds. Each round has approximately the same overall budget but differs in the number of configurations and fidelity. For example, using the default settings of $B = 3$, $n = 64$, and $\eta = 4$, GraSH uses three rounds with 64, 16, and 4 configurations and a fidelity of 1/64, 1/16, and 1/4, respectively. The hyperparameter configurations in the first round are sampled randomly from the hyperparameter space. Depending on the variant being used, GraSH reduces the number of epochs, the graph size, or both to reach the desired fidelity. If no graph with size corresponding exactly to the budget defined by

the fidelity exists, the next smaller one is used. After validating each configuration (see below), the best-performing $1/\eta$-th of the configurations is passed on to the next round. This process is repeated until only one configuration remains.

Validation on subgraphs. Care must be taken when validating a KGE model trained on a subgraph, e.g., $\mathcal{G}_i = (\mathcal{E}_i, \mathcal{R}_i, \mathcal{K}_i)$ in round i. The full graph $\mathcal{G}$ comes with training G^{train}, validation G^{valid} and test split G^{test}. Since $\mathcal{G}_i$ typically contains a reduced set of entities $\mathcal{E}_i \subseteq \mathcal{E}$, the full validation set G^{valid} cannot be used without ignoring all triples containing entities $\mathcal{E} \setminus \mathcal{E}_i$. This is because no embedding is learned for the "unseen" entities in $\mathcal{E} \setminus \mathcal{E}_i$, so that we cannot score any triples containing these entities (as required by the entity ranking protocol). While this filtering procedure offers a comparable validation score to the full validation set for graphs containing close to all entities, it might lead to small validation sets for subgraphs with a small set $\mathcal{E}_i$ and might introduce a query selection bias depending on the graph reduction technique. To avoid this problem, we explicitly create new train and valid splits $\mathcal{G}_i^{\text{train}}$ and $\mathcal{G}_i^{\text{valid}}$ in round i. Here, $\mathcal{K}_i^{\text{valid}}$ is sampled randomly from $\mathcal{K}_i$ and $\mathcal{K}_i^{\text{train}} = \mathcal{K}_i \setminus \mathcal{K}_i^{\text{valid}}$. Although this approach is very simple, it worked well in our study. An alternative is the construction of "hard" validation sets, e.g., as proposed by Toutanova and Chen (2015). We leave the exploration of such techniques to future work.

Negative sample scaling. Recall that the number N^- of negative samples is an important hyperparameter for KGE model training. Generally (and assuming without-replacement sampling), each entity is sampled as a negative with probability $N^-/|\mathcal{E}|$. When we use a subgraph $\mathcal{G}_i$ as in GraSH, this probability increases to $N^-/|\mathcal{E}_i|$, i.e., each entity is more likely to act as a negative sample due to the reduction of the number of entities. To correctly assess hyperparameter configurations in such cases, GraSH scales the number of negative examples and uses $N_i^- = \frac{|\mathcal{E}_i|}{|\mathcal{E}|} N^-$ in round i. This choice preserves the probability of sampling each entity as a negative and provides additional cost savings since the total number of scored triples is further reduced in low-fidelity experiments.

Cost model and budget allocation. To distribute the search budget B over the rounds, we make use of a simple cost model to estimate the relative runtime of low-fidelity approximations. This cost model drives the choice of f_i in Alg. 4. In particular, we assume that training cost is linear in both the number of epochs (E_i) and the number of triples ($|\mathcal{K}_i|$). For example, this implies that training five configurations for one epoch has the same cost as training one configuration for five epochs. Likewise, training five configurations with 20% of the triples has the same cost as training one configuration on the whole KG. Using this assumption, the relative cost of evaluating a single hyperparameter configuration in round i is given by $\frac{E_i}{E} \frac{|\mathcal{K}_i|}{|\mathcal{K}|}$. More elaborate

cost models are conceivable, but this simple approach already worked well in our experimental study. Note, for example, that our simple cost model neglects negative sample scaling and thus tends to overestimate (but avoids underestimation of) training cost.[b]

4.4 Low-fidelity Approximation Techniques

In this section, we summarize and discuss various low-fidelity approximation techniques. As discussed previously, the two most common types are *graph reduction* (i.e. training on a reduced graph) and *epoch reduction* (i.e., training for fewer epochs). Note that although graph reduction is related to dataset reduction techniques used in other machine learning domains, it represents a major challenge since the relationships between entities need to be taken into account.

Generally, good low-fidelity approximations satisfy the following criteria:

1) **Low cost.** Computational and memory costs for model training (including model initialization) and evaluation should be low. Recall that computational cost is mainly determined by the number of triples, whereas memory and evaluation cost are determined by the number of entities. Ideally, both quantities are reduced.
2) **High transferability.** Low-fidelity approximations should transfer to the full KG in that they provide useful information. E.g., rankings of hyperparameter configurations evaluated using low-fidelity approximations should match or correlate with the rankings at full-fidelity.
3) **Flexibility.** It should be possible to flexibly trade off computational cost and transferability.

All three points are essential for cost-effective and practical multi-fidelity HPO.

In the following, we present the graph reduction approaches *triple sampling*, *entity sampling*, *multi-start random walk*, and *k-core decomposition*, as well as epoch reduction. A high-level comparison of these approaches w.r.t. the above desiderata is provided in Tab. 4.1. The assessment given in the table is based on our experimental results (Sec. 4.5.2).

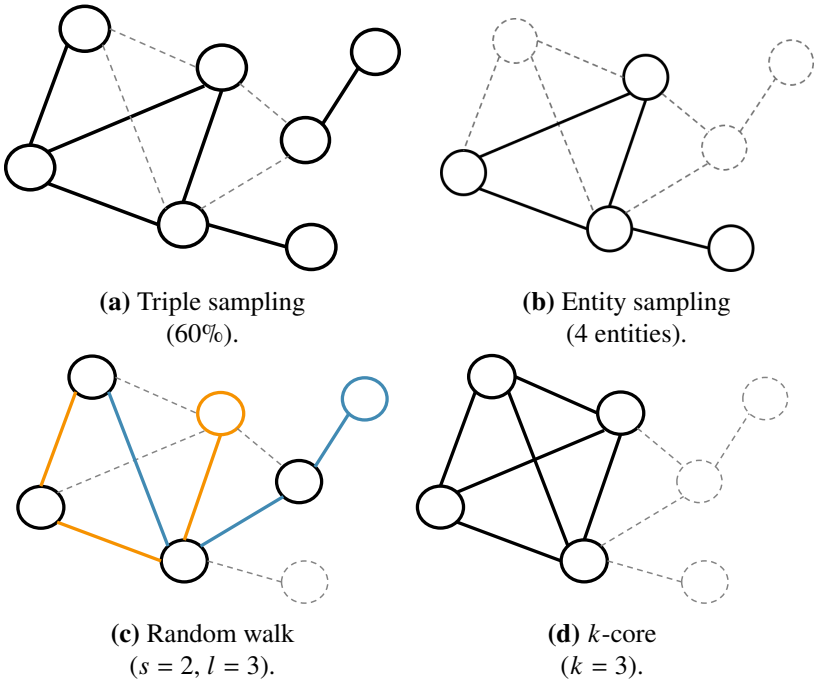
4.4.1 Graph Reduction

Graph reduction techniques produce a reduced KG $\mathcal{G}_i = (\mathcal{E}_i, \mathcal{R}_i, \mathcal{K}_i)$ from the full KG $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{K})$. This is commonly done by first determining the reduced set $\mathcal{K}_i$

[b] Accounting for the negative sampling scaling, the cost model would result in $\frac{E_i}{E}\frac{|\mathcal{K}_i|}{|\mathcal{K}|}(1+\frac{|\mathcal{E}_i|}{|\mathcal{E}|}N^-)$.

Technique	Low Cost	High Transferability	Flexibility
Triple sampling	○	-	+
Entity sampling	○	○	+
Random walk	○	○	+
k-core decomposition	+	+	○
Epoch reduction	-	○	+

Table 4.1: Comparison of low-fidelity approximation techniques.



(a) Triple sampling (60%).

(b) Entity sampling (4 entities).

(c) Random walk ($s = 2, l = 3$).

(d) k-core ($k = 3$).

Figure 4.3: Schematic illustration of selected graph reduction techniques. All reduced graphs contain 6 of the 10 original triples but a varying number of entities.

of triples and subsequently retaining only those entities (in $\mathcal{E}_i$) and relations (in $\mathcal{R}_i$) that occur in $\mathcal{K}_i$.[c] A reduction in triples thus may lead to a reduction in the number of entities and relations as well. This consequently results in further savings in computational cost, evaluation cost, and memory consumption. The graph reduction techniques discussed here are illustrated in Fig. 4.3.

Triple sampling (Fig. 4.3a). The perhaps simplest approach to reduce graph size is to sample triples randomly from the graph. As shown in Fig. 4.3a, many entities with sparse interconnections can remain in the resulting subgraphs (e.g., the two entities at the top right) so that $\mathcal{E}_i$ tends to be large and $\mathcal{G}_i$ consists of many connected

[c] All other entities/relations do not occur in the reduced training data so we cannot learn useful embeddings for them.

Graph reduction	Triples	Entities	Median entity degree	Connected components
Triple sampling	10 790	15 296	1	4 540
Entity sampling	8 512	6 298	2	743
Random walk	10 790	11 178	1	1 967
k-core	10 179	317	53	1

Table 4.2: Statistics of a Yago3-10 subgraph with $\approx 1\%$ of triples.

components. See Tab. 4.2 for subgraph statistics. The cost in terms of model size and evaluation time is consequently only slightly reduced. We also observed (see Sec. 4.5.2) that triple sampling leads to low transferability, most likely due to this sparsity. Triple sampling does offer very good flexibility, however, since triple sets of any size can be constructed easily.

Entity sampling (Fig. 4.3b). Instead of sampling the triples directly, a subset of entities can be sampled uniformly at random to reduce the graph. Here, the resulting graph $\mathcal{G}_i$ is the induced graph of all sampled entities $\mathcal{E}_i$. Sampled entities without any connection in the induced graph are discarded. This induced graph leads to a considerably better-connected graph with fewer entities and connected components compared to triple sampling; see Tab. 4.2. We observed that entity sampling leads to improved transferability over triple sampling. However, flexibility is reduced. Due to the combination of skewed entity degrees and random sampling of entities, the size of an induced graph is hard to estimate, and finding an induced graph of expected size can be an exhaustive search.

Random walk (Fig. 4.3c). In multi-start random walk, which is used in AutoNE (Tu et al. 2019), a set of s random entities is sampled from $\mathcal{E}$. A random walk of length l is started from each of these entities and the resulting triples form $\mathcal{K}_i$. Empirically, many entities may ultimately remain so that the reduction of memory consumption and evaluation cost is limited. Although the resulting subgraph tends to be better connected than the ones obtained by triple sampling (c.f. Tab. 4.2), transferability is still low and close to triple sampling (again, see Sec. 4.5.2). As triple sampling, the approach is very flexible though. KGTuner (Zhang et al. 2022) improves on the basic random walk considered here by using biased starts and adding all connections between the retained entities (even if they do not occur in a walk); i.e., constructing the induced graph, as done in entity sampling. The k-core decomposition, which we discuss next, offers a more direct approach to obtaining such a highly connected graph.

k-core decomposition (Fig. 4.3d). The k-core decomposition (Seidman 1983) allows for the construction of subgraphs with increasing cohesion. The k-core subgraph of $\mathcal{G}$, where $k \in \mathbb{N}$ is a parameter, is defined as the largest induced subgraph in which every retained entity (i.e., $\mathcal{E}_i$) occurs in at least k retained triples (i.e., in $\mathcal{K}_i$). The computation of k-cores is cheap and supported by common graph libraries. Generally, the number of entities contained in G_i is reduced drastically with increasing k, resulting in considerably fewer entities compared to triple sampling and random walk (c.f. Tab. 4.2). This is due to the skewed degree distribution of entities present for most graphs; i.e., few high-degree and many low-degree entities. For k-cores with $k > 1$ long-tail entities with infrequent connections are removed. Moreover, they are highly interconnected by construction. As a consequence, we found that computational cost and memory consumption are low and transferability high. The approach is less flexible than the other graph reduction techniques, as the choice of k and the graph structure determines the resulting fidelity. One may interpolate between k-cores for improved flexibility, but we did not explore this approach in this work.

4.4.2 Epoch Reduction

Epoch reduction is the most common form of fidelity control used in HPO (Baker et al. 2018; Wang et al. 2021a). As the set $\mathcal{E}$ of entities does not change with varying fidelity, memory and evaluation cost are very large even when using low-fidelity approximations. We observed good transferability as long as the number of epochs is not too small (Sec. 4.5.2); otherwise, transferability is often considerably worse than graph reduction techniques. This limits flexibility: Especially on large-scale graphs, the overall training budget often consists of only a small number of epochs in the first place (e.g., 10 as in Ch. 3 and Zheng et al. (2020)). Note that the available budget in low-fidelity approximations can be smaller than the cost of one complete epoch (when $f_i < 1/E$ in Alg. 4). Although partial epochs can be used easily, epoch reduction then corresponds to a form of triple sampling (with the additional disadvantage of not reducing the set of entities).

4.4.3 Summary

In summary, as long as the desired fidelity is sufficiently high, epoch reduction offers high-quality approximations and high flexibility. It does not improve memory consumption and evaluation cost, however, and it leads to high cost and low quality on large-scale graphs with limited budget. Graph reduction approaches, on the other hand, reduce the number of entities and hence memory consumption and evaluation cost. Compared to triple sampling, entity sampling, and random walks, the k-core

decomposition has the highest transferability and lowest cost. In GraSH, we use a combination of epoch reduction and k-core decomposition by default to avoid training for partial epochs and the use of very small subgraphs with low fidelity.

4.5 Experimental Study

We conducted an experimental study to answer the following research questions: (i) To what extent can low-fidelity approximations be used for hyperparameter evaluation for the task of link prediction in KGs? (ii) Can the multi-fidelity search algorithm GraSH improve upon using low-fidelity approximations only and reach quality close to a full search and current state-of-the-art? To address question (i), we investigated to what extent hyperparameter rankings obtained with low-fidelity approximations correlate with the ones obtained at full fidelity (Sec. 4.5.2). For question (ii), we used the best low-fidelity approximations in combination with GraSH, and evaluated the resulting model quality (Sec. 4.5.3), resource consumption during HPO (Sec. 4.5.4), and the robustness of the search algorithm (Sec. 4.5.5). Further, we compared GraSH to the parallel work of KGTuner (Zhang et al. 2022) (Sec. 4.5.3). In summary, we found that:

1) GraSH was cost-effective and produced high-quality hyperparameter configurations. It reached state-of-the-art results on a large-scale graph with a small overall search budget of three complete training runs (Sec. 4.5.3).
2) Using multiple reduction techniques was beneficial. In particular, a combination of graph- and epoch-reduction performed best (Sec. 4.5.2 and 4.5.3).
3) Low-fidelity approximations correlated best to full fidelity for graph reduction using the k-core decomposition and, as long as the budget was sufficiently large, second-best for epoch reduction (Sec. 4.5.2).
4) Graph reduction was more effective than epoch reduction in terms of reducing computational and memory cost. Evaluation using small subgraphs had low memory consumption and short runtimes (Sec. 4.5.4).
5) Using multiple rounds with increasing fidelity levels was beneficial (Sec. 4.5.5).
6) GraSH was robust to changes in budget allocation across rounds (Sec. 4.5.5).

4.5.1 Experimental Setup

Source code, search configurations, and resulting hyperparameters can be found at `https://github.com/uma-pi1/GraSH`.

Scale	Dataset	Entities	Relations	|Train|	|Valid|	|Test|
Small	Yago3-10	123 182	37	1 079 040	5 000	5 000
Medium	Wikidata5M	4 594 485	822	21 343 681	5 357	5 321
Large	Freebase	86 054 151	14 824	304 727 650	1 000	10 000

Table 4.3: Dataset used in this study.

Datasets. We used common KG benchmark datasets of varying sizes with a focus on larger datasets; *Yago3-10* (Dettmers et al. 2018), *Wikidata5M* (Wang et al. 2021b), and the largest dataset *Freebase* (Zheng et al. 2020). For statistics see Tab. 4.3 and for details Sec. 2.6. For all datasets except Freebase, we use the validation and test sets that accompany the datasets to evaluate the final model. For Freebase, we used the same sub-sampled validation (1 000 triples) and test sets (10 000 triples) from Ch. 3.[d]

Hardware. All runtime, GPU memory, and model size measurements were taken on the same machine (40 Intel Xeon E5-2640 v4 CPUs @ 2.4GHz; 4 NVIDIA GeForce RTX 2080 Ti GPUs).

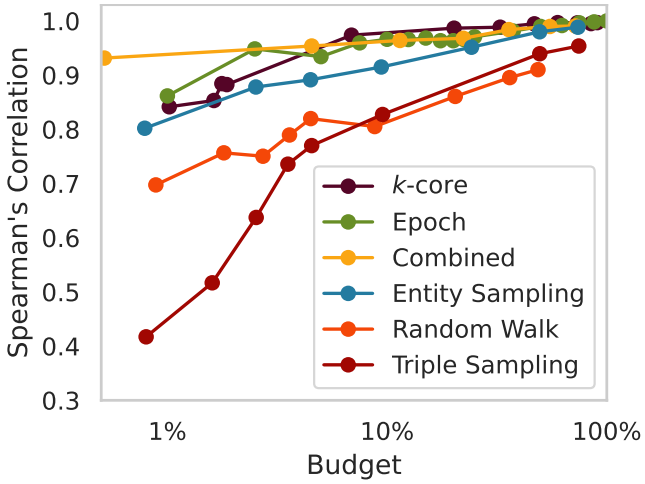
Implementation and models. GraSH uses DistKGE, presented in Ch. 3, for parallel training of large-scale graphs and HpBandSter (Falkner et al. 2018) for the implementation of SH. We considered the models ComplEx (Trouillon et al. 2016), RotatE (Sun et al. 2019), and TransE (Bordes et al. 2013). All three models are commonly used for large-scale KGEs (Lerer et al. 2019; Zheng et al. 2020).

Hyperparameters. We used the same hyperparameter search space as in Ch. 3. The search space consists of nine continuous and two categorical hyperparameters. The upper bound on the number of negative samples for ComplEx is 10 000 and for RotatE and TransE 1 000 (since these models are more memory-hungry). We set the maximum training epochs on Yago3-10 to 400, on Wikidata5M to 64, and on Freebase to 10.

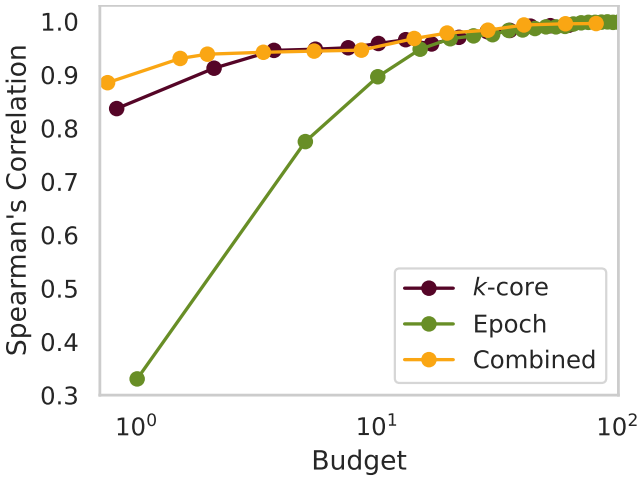
Methodology. For the GraSH search, we used the default settings ($B = 3$, $\eta = 4$, $n = 64$). Apart from the upper bound of negatives, we used the same 64 initial hyperparameter settings for all models and datasets to allow for a fair comparison. For graph reduction, we used k-core decomposition unless mentioned otherwise. Subgraph validation sets generated by GraSH consisted of 5 000 triples.

Metrics. We used the common filtered MRR metric to evaluate KGE model quality on the link prediction task as described in Sec. 2.5. Additionally, results for Hits@k are presented in Tab. A.1 in Appendix A.

[d] The original test set contains ≈17 M triples, which leads to excessive evaluation costs. For the purpose of MRR computation, a much smaller test set is sufficient.



(a) Yago3-10 (max. 40 epochs).



(b) Wikidata5M (max. 20 epochs).

Figure 4.4: Comparison of low-fidelity approximation techniques. Shows Spearman's rank correlation between low-fidelity approximations and a full-fidelity baseline. Budget (log-scale) corresponds to the relative number of epochs and/or triples.

4.5.2 Comparison of Low-Fidelity Approximation Techniques (Fig. 4.4)

To address the question of to what extent low-fidelity approximation can be used for hyperparameter evaluation, we studied and compared the transferability of low-fidelity approximations to full-fidelity results. To do so, we first ran a full-fidelity hyperparameter search consisting of 30 pseudo-randomly generated trials. We then trained and evaluated the same 30 trials using the approximation techniques described in Sec. 4.4 at various budgets. To keep computational cost feasible, this experiment was only performed on the two smaller datasets and with a smaller number of epochs.

Since the validation sets used with graph reductions differ from the ones used at full fidelity (see Sec. 4.3), we compared the ranking of hyperparameter configurations instead of their MRR metrics. In particular, we used Spearman's rank correlation coefficient (Zwillinger and Kokoska 1999) between the low-fidelity and the high-fidelity results. A higher value corresponds to a better correlation.

Our results on Yago3-10 are visualized in Fig. 4.4a. We found high transferability for the k-core decomposition and epoch reduction. Graph reduction based on triple sampling and random walks led to clearly inferior results and was not further considered. A combination of k-core subgraphs and reduced epochs (each contributing 50% to the savings) further improved low-budget results.

To investigate the behavior on a larger graph, we evaluated the three best techniques on Wikidata5M, see Fig. 4.4b. Recall that due to the high cost, a small number of epochs is often used for training on large KGs. This has a detrimental effect on the transferability of epoch reduction, as partial epochs need to be used for low-fidelity approximations (see Sec. 4.4.2). In particular, there is a considerable drop in transferability for epoch reduction below a budget of 10%. This drop in performance is neither visible for the k-core approximations nor for the combined approach.

Note that even for the best low-fidelity approximation, the rank correlation increased with budget. This suggests that using multiple fidelities (as in GraSH) instead of a single fidelity is beneficial. In our study, this was indeed the case (see Sec. 4.5.5).

4.5.3 Final Model Quality (Tab. 4.4)

To analyze the performance of GraSH in terms of the quality of its selected hyperparameter configurations, we evaluated the search approach on three datasets, using three models with varying model dimensionalities. We used GraSH with epoch reduction, graph reduction, as well as the combined approach. Test-data performance of the resulting configurations trained at full fidelity is summarized in Tab. 4.4a.

Results (Tab. 4.4a). The combined variant of GraSH offered best or close to best results across all datasets and models. In comparison to the other variants, it avoided the drawbacks of training partial epochs (e.g., epoch reduction on Freebase) as well as using subgraphs that are too small (e.g., graph reduction on Yago3-10).

Comparison to prior results (Tab. 4.4b). We compared the results obtained by GraSH to the best published prior results known to us, see Tab. 4.4b. Note that prior models were often trained at substantially higher cost. For example, on Wikidata5M, GraSH used an overall budget of $4 \cdot 64 = 256$ epochs for HPO and training, whereas some prior methods used 1 000 epochs for a single training run. Likewise, dimen-

	Dataset	Reduc- → tion Model ↓	Epoch Dim 128	Graph Dim 128	Comb. Dim 128	Comb. Dim 512
Small	Yago 3-10 ($E = 400$)	ComplEx	**0.536**	0.463	0.528	0.552
		RotatE	0.432	0.432	**0.434**	0.453[e]
		TransE	**0.499**	0.422	**0.499**	0.496
Medium	Wiki-data5M ($E = 64$)	ComplEx	**0.300**	**0.300**	**0.300**	0.294
		RotatE	**0.241**	0.232	**0.241**	0.261
		TransE	0.263	0.263	**0.268**	0.249
Large	Free-base ($E = 10$)	ComplEx	0.572	**0.594**	**0.594**	0.678
		RotatE	0.561	**0.613**	**0.613**	0.615
		TransE	0.261	**0.553**	**0.553**	0.559

(a) GraSH with default settings ($B = 3$, $n = 64$, $\eta = 4$).

MRR	Dim	Epochs	
0.551	128	400	†[f]
0.495	1 000	?	††
0.510[g]	350	4 000	‡
0.308	128	300	‡‡
0.290	512	1 000	$
0.253	512	1 000	$
0.612	400	10	‡‡
0.567	128	10	‡‡
-	-	-	

(b) Prior results

Table 4.4: Model quality in terms of MRR. State-of-the-art results underlined. Best reduction variant in bold. Note that best prior results often use a considerably larger budget and/or model dimensionality. Marked results are from † Broscheit et al. (2020), †† Sun et al. (2019), ‡ Costabello et al. (2019), ‡‡ Chap 3, $ Wang et al. (2021b)

sionalities of up to 1 000 were sometimes used. For a slightly more informative comparison, we performed a GraSH search with an increased dimensionality of 512, but kept the low search and training budgets. Even with this low budget, we found that on small to midsize graphs, GraSH performed either similarly (ComplEx, Yago3-10 & Wikidata5M) or sometimes slightly worse (RotatE, Wikidata5M) than the best prior results. On the large-scale Freebase KG, where low-fidelity hyperparameter search is a necessity, GraSH outperformed state-of-the-art results by a large margin.

Comparison to KGTuner. KGTuner (Zhang et al. 2022) was developed in parallel to this work and follows similar goals as GraSH. We compared the two approaches on the smaller Yago3-10 KG with ComplEx; a comparison on the larger datasets was not feasible since KGTuner has large computational costs. We ran both GraSH and KGTuner with the default settings of KGTuner ($n = 50$ trials, $E = 50$ epochs, dim. 1 000) to obtain a fair comparison. KGTuner reached an MRR of 0.505 in about 5 days (its search budget corresponds to $B \approx 20$). GraSH reached an MRR of 0.530 in about 1.5 hours ($B = 3$, sequential search on 1 GPU), i.e., a higher quality result at lower cost. The high computational cost of KGTuner mainly stems from its inflexible

[e] RotatE benefits from self-adversarial sampling as used in (Sun et al. 2019). We did not use this technique to keep the search space consistent across all models. An adapted GraSH search space led to an MRR of 0.494 (combined, $d = 512$), matching the prior result.

[f] Published in the online appendix of (Broscheit et al. 2020).

[g] Published with the AmpliGraph library (Costabello et al. 2019), which ignores unseen entities during evaluation. This inflates the MRR so that results are not directly comparable.

		Round time (min)			Model size (MB)		
		Epoch	Graph	Comb.	Epoch	Graph	Comb.
Yago3-10	Round 1	43.9	24.7	15.9	60.2	0.3	2.0
	Round 2	34.8	13.3	27.1	60.2	0.4	6.3
	Round 3	38.7	28.1	33.5	60.2	6.3	16.7
	Total	117.4	**66.1**	76.5			
Wikidata5M	Round 1	182.3	60.1	82.3	2 353.3	1.0	71.3
	Round 2	134.2	87.4	88.6	2 353.3	36.0	182.0
	Round 3	126.9	92.5	95.3	2 353.3	182.0	454.7
	Total	443.4	**240.0**	266.2			
Freebase	Round 1	915.9	250.7	179.7	42 025.9	87.3	1 322.2
	Round 2	507.9	172.0	151.2	42 025.9	520.1	2 667.7
	Round 3	423.4	197.5	207.0	42 025.9	2 667.7	6 571.3
	Total	1 847.2	620.2	**537.9**			

Table 4.5: Resource consumption per round (ComplEx).[h]

and inefficient budget allocation (e.g., always 10 full-fidelity evaluations). The higher quality of GraSH stems from its use of multiple fidelities (vs. two in KGTuner) and by using a combination of k-cores and epoch reduction (vs. random walks in KGTuner).

4.5.4 Resource Consumption (Tab. 4.5)

Next, we investigated the computational cost and memory consumption of each round of GraSH. We used 4 GPUs in parallel, evaluating one trial per GPU with the same settings as used in Sec. 4.5.3. Our results are summarized in Tab. 4.5.

Memory consumption. Epoch reduction was less effective than graph reduction and a combined approach in terms of memory usage. With epoch reduction, training is performed on the full graph in every round and therefore performed with full model size. Due to the large model sizes on the largest graph Freebase, the model could not be kept in GPU memory introducing further overheads for parameter management (for more details, see Ch. 3). Graph reduction with k-core decomposition reduced the number of entities contained in a subgraph considerably. As the model size is mainly driven by the number of entities, the resulting model sizes were small.

Runtime. Similarly to memory consumption, a GraSH search based on epoch reduction was less effective in terms of runtime compared to graph reduction and a combined approach. With epoch reduction, runtime was mainly driven by the cost

[h]The time needed to compute the k-core decompositions is excluded. It is negligible compared to the overall search time (e.g., ≈ 28 min for Freebase with igraph (Csardi et al. 2006)).

Dataset	$\eta = 2$ 6 rounds $B = 3$	$\eta = 4$ 3 rounds $B = 3$ (default)	$\eta = 8$ 2 rounds $B = 3$	$\eta = 64$ 1 round $B = 3$	$\eta = 64$ 1 round $B = 1$
Yago3-10	0.463	0.463	0.485	0.427	0.427
Wikidata5M	0.300	0.300	0.300	0.300	0.285
Freebase	0.594	0.594	0.594	0.572	0.572

Table 4.6: Influence of the number of rounds on model quality in terms of MRR (ComplEx, graph reduction, $n = 64$ trials, $B = 3$). The number of rounds is directly controlled by the choice of n and η.

of model evaluation and model initialization. This is especially visible in the first round of the search on large graphs. Here, the number of trials and therefore the number of model initializations and evaluations is high. Additionally, on the largest graph, the overhead for parameter management for training on the full KG increased runtime further. In contrast, small model sizes and low GPU utilization with graph reduction would even allow further performance gains. For example, improving on the presented results, the runtime of the first round on Wikidata5M can be reduced from 60.1 to 22.9 minutes by training three models per GPU instead of one.

4.5.5 Influence of Number of Rounds (Tab. 4.6)

In our final experiment, we address the question of whether we need multiple fidelities, as well as how sensitive GraSH is to the number of rounds (i.e. the number of fidelities) being used. Our results are summarized in Tab. 4.6. All experiments were conducted at the same budget ($B = 3$) and number of trials ($n = 64$). Note that the number of rounds used by GraSH is given by $\log_\eta(n)$, where n denotes the number of trials and η the reduction factor. The smaller η, the more rounds are used and the lower the (initial) fidelity.

We found that on the two larger graphs, the search was robust to changes in budget allocation and η did not influence the final trial selection (as long as at least 2 rounds were used). Only on the smaller Yago3-10 KG, the final model quality differed with varying values of η. Here, low-fidelity approximation (small η) was riskier since the subgraphs used in the first rounds were very small.

To investigate whether multi-fidelity HPO—i.e., multiple rounds—are beneficial, we (i) used the best configuration of the first round directly ($\eta = 64$, $B = 1$) and (ii) performed an additional single-round search with a comparable budget to all other settings ($\eta = 64$, $B = 3$). As shown in Tab. 4.6, both settings did not reach

the performance achieved via multiple rounds. We conclude that the use of multiple fidelity levels is essential for cost-effective HPO.

4.6 Conclusion

We first presented and experimentally explored various low-fidelity approximation techniques for evaluating hyperparameters of KGE models. Based on our findings, we proposed GraSH, an open-source, multi-fidelity hyperparameter optimizer for KGE models based on successive halving. We found that GraSH often reproduced or outperformed state-of-the-art results on large knowledge graphs at very low overall cost, i.e., the cost of three complete training runs. We argued that the choice of low-fidelity approximation is crucial (k-core reduction combined with epoch reduction worked best), as is the use of multiple fidelities.

4.6.1 Limitations

Finding k-core subgraphs corresponding to a given fidelity is dependent on the underlying graph structure. For some graphs with a highly skewed entity degree, one might not be able to find a fitting subgraph, i.e., the subgraph might be too big or too small. While interpolating between k-cores can increase flexibility, in extreme cases, this approach reduces to entity sampling.

Further, k-core decomposition introduces a bias to keep high-degree entities. While this approach worked well on general domain KG evaluated in this study, the transferability of such cores might be less for some domain-specific graphs.

Finally, our cost model for subgraph selection only considered training cost (selection by number of triples) and ignored evaluation cost. However, the entity ranking protocol used for evaluation comes with considerable cost which scales linearly with the number of entities contained in the graph. Taking this cost into account could change the subgraph selection and in turn resulting quality and resource consumption. Nevertheless, parts of the cost reduction of GraSH compared to other HPO approaches come from the cheaper evaluation on subgraphs. For a more detailed comparison, future work could compare GraSH to HPO approaches utilizing more cost-efficient evaluation strategies. E.g., integrating maximum inner product search (Aguerrebere et al. 2023; Douze et al. 2024) into entity ranking as discussed in Sec. 2.5.

4.6.2 Future Work

Bayesian optimization. GraSH builds upon successive halving. BOHB (Falkner et al. 2018) integrates Bayesian optimization into the successive halving extension Hyperband (Li et al. 2017). This showed promising results on multiple HPO benchmarks. A similar Bayesian optimization integration could further strengthen the resulting quality of GraSH.

HPO for GNNs. Recently GNNs showed promising results for link prediction in KGs (Zhu et al. 2021). However, approximating the performance of hyperparameter settings might work worse for GNNs, as representations are strongly dependent on the graph structure and size of the graph. Further research is needed to evaluate the implications and performance of HPO on subgraphs for GNNs.

Reuse subgraph representations. Currently, in each round GraSH trains models from scratch if subgraph size increases. Reusing parameters trained on smaller graphs by initializing corresponding parameters in the next larger model could further decrease cost. However, first experiments in this regard showed a decrease in final model quality. Further research could explore necessary changes in parameter updates to keep quality at a high level and make use of pretrained subgraph representations.

Chapter 5

Sequence-to-Sequence Link Prediction

"If words are not things, or maps are not the actual territory, then, obviously, the only possible link between the objective world and the linguistic world is found in structure, and structure alone."

Alfred Korzybski, 1958

While relying on graph structure only, KGEs offer high link prediction quality. They can be trained (Ch. 3) and tuned (Ch. 4) efficiently and effectively. However, next to the structural, KGs often carry textual information, e.g., entity mentions and descriptions. A proper integration of both information sources can further improve link prediction and downstream task performance. In this chapter, we show that an off-the-shelf encoder-decoder Transformer model can serve as a scalable and versatile model obtaining state-of-the-art results for link prediction. We achieve this by posing link prediction as a sequence-to-sequence task and exchanging the triple-scoring approach taken by prior KGE methods with autoregressive decoding. Such a simple but powerful method reduces the model size up to 98% compared to conventional KGE models while matching or even improving upon state-of-the-art link prediction approaches.

This chapter is based on Saxena et al. (2022) and Kochsiek et al. (2023).

5.1 Introduction

KGE models can be trained (Ch. 3) and tuned (Ch. 4) efficiently and effectively on large-scale graphs. They offer high-quality link prediction models relying on graph structure only. However, text-attributed graphs are prevalent in many real-world scenarios (Yan et al. 2023), and integration of this textual information can further benefit the resulting quality. Especially KGs carry valuable textual information, such as entity mentions and descriptions, for example, mapped from Wikipedia or domain-specific sources. Given this textual information, and taking into account the large size of real-world KGs (Wikidata contains ≈90M entities), as well as the applicability to downstream tasks, link prediction models should be able to utilize the textual information effectively while fulfilling the following desiderata:

(i) **Scalability.** Have model size and inference time independent of the number of entities,
(ii) **Quality.** Reach good empirical performance,
(iii) **Versatility.** Be applicable for multiple tasks such as link prediction and question answering,
(iv) **Simplicity.** Consist of a single module with a standard architecture and training pipeline.

Existing approaches integrating textual information lack at least one of these desiderata. They either fall behind the state-of-the-art in link prediction quality (Clouatre et al. 2021; Yao et al. 2019; Wang et al. 2021b; Xie et al. 2016), introduce high cost limiting scalability (Yao et al. 2019), or are limited in terms of versatility (Wang et al. 2021b, 2022; Xie et al. 2016). A comparison of approaches in terms of desiderata is summarized in Tab. 5.1 with a detailed overview in Sec. 5.2.

In this chapter, we show that all of these desiderata can be fulfilled by a simple sequence-to-sequence (seq2seq) model. To this end, we pose KG link prediction as a seq2seq task and train an encoder-decoder Transformer model (Vaswani et al. 2017) on this task. This simple but powerful approach, which we call KGT5, is visualized in Fig. 5.1. With such a unified seq2seq approach, we achieve (i) scalability – by using compositional entity representations and autoregressive decoding (rather than scoring all entities) for inference, (ii) quality – we obtain state-of-the-art performance, (iii) versatility – the same model can be used for both link prediction and question answering over incomplete KGs, and (iv) simplicity – we obtain all results using an off-the-shelf model with no task or dataset-specific hyperparameter tuning.

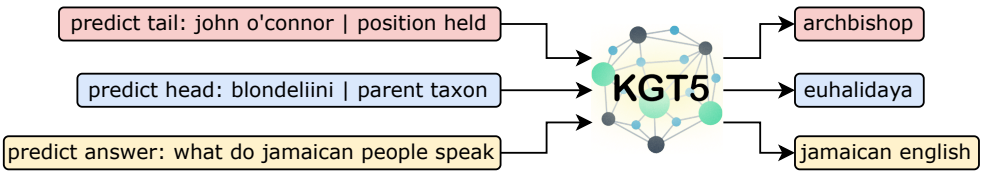


Figure 5.1: Overview of our method KGT5. KGT5 is first trained on the link prediction task (predicting head/tail entities, given tail/head and relation). For question answering, the same model is further finetuned using QA pairs.

KGT5 offers a strong integration of structural and textual information. General themes for such an integration of KGs and language models are a KG lookup during inference (Lewis et al. 2020; Li et al. 2023; Wu et al. 2022), or a KG integration during pretraining (Kang et al. 2022; Li et al. 2022b; Xiong et al. 2020). KGT5 is a combination of both. It is trained directly on the KG itself capturing textual and structural information in a unified manner. With an additional graph lookup during inference, KGT5's extension—KGT5-context—offers further improvements in terms of link prediction quality.

In summary, we make the following contributions:

- We show that KG link prediction can be treated as sequence-to-sequence tasks and tackled successfully with a single encoder-decoder Transformer (with the same architecture as T5 (Raffel et al. 2020)).
- With this simple but powerful approach called KGT5, we reduce model size for KG link prediction up to 98% while outperforming conventional KGEs on a dataset with 90M entities.
- We show that the link prediction performance of such a seq2seq approach can be strengthened significantly via a simple graph lookup, i.e., integrating the query entity's one-hop neighborhood.

Further, building upon these contributions, Saxena et al. (2022) showcase the versatility of this approach through the task of question answering over incomplete KGs. By pretraining on KG link prediction and finetuning on question answering, KGT5 performs similar to or better than much more complex methods on multiple large-scale benchmarks.

5.2 Related Work

Previously, and in parallel to this work, multiple approaches utilizing textual information for link prediction were presented. However, existing approaches lack at least one of the desiderata presented in Sec. 5.1. We first compare related work in terms

	(Sc)	(Q)	(V)	(Si)	Pre-trained	Text-model	Approach
KGE		✓		✓		-	bi-encoder
HitteER (Chen et al. 2021)		✓				-	hierarchical bi-encoder
KGE + MPNet (Hu et al. 2021)		✓		✓		MPNet	bi-encoder
DKRL (Xie et al. 2016)	✓			✓	✓	word2vec + CNN	bi-encoder
BLP (Daza et al. 2021)	✓			✓	✓	BERT	bi-encoder
KEPLER (Wang et al. 2021b)	✓			✓	✓	RoBERTa	bi-encoder + MLM
SimKGC (Wang et al. 2022)	✓	✓		✓	✓	BERT	bi-encoder
KG-BERT (Yao et al. 2019)			✓	✓	✓	BERT	cross-encoder
MLMLM (Clouatre et al. 2021)	✓		✓	✓	✓	RoBERTa	encoder + sampling
Better Together[a] (Chepurova et al. 2023)	✓	✓	✓	✓	✓	T5	encoder-decoder
KGT5 (ours)	✓	✓	✓	✓		T5	encoder-decoder
KGT5-context (ours)	✓	✓	✓	✓		T5	encoder-decoder

Table 5.1: Comparison of related work in terms of the desiderata described in Sec. 5.1, grouped by approach with the first group showcasing non-text models.
(Sc) - Scalability, (Q) - Quality, (V) - Versatility, (Si) - Simplicity.

of textual integration for the link prediction alongside presented desiderata, followed by a summary of approaches integrating neighborhood information. For an overview, see Tab. 5.1.

5.2.1 Text-Based Link Prediction Models

Multiple textual link prediction approaches evolved from the KGE architecture. Due to the non-parametric nature of KGEs scaling linearly with the number of entities in the graph, KGEs result in a high parameter count limiting their scalability. Additionally, they lack versatility. While, for example, question-answering methods leveraging KGEs outperform traditional approaches on incomplete KGs, combining KGEs with the question-answering pipeline is a non-trivial task (Huang et al. 2019; Ren et al. 2021; Sun et al. 2021; Saxena et al. 2020). Textual integrations can help address

[a]Better Together (Chepurova et al. 2023) was published after this study, and shows with a very similar approach to KGT5-context similar improvements.

these drawbacks. E.g., by allowing for compositional entity embeddings addressing the scalability issue, making versatile adaptions to downstream applications possible, and further improving link prediction quality. However, seldom all drawbacks are addressed at once. We grouped textual link prediction approaches by the architecture types introduced in Sec. 2.3, starting with simple baselines integrating text directly into the KGE architecture.

Integrating text into KGEs. A baseline to utilize textual together with structural information is to concatenate learnable KGEs with text embeddings of entity mentions and descriptions; e.g., Hu et al. (2021) use MPNet (Song et al. 2020) to embed textual information. With useful textual information, this baseline can improve quality but comes with the limited scalability and versatility of KGEs.

Bi-encoder. Addressing the limited scalability, DKRL (Xie et al. 2016) and BLP Daza et al. (2021) do not use learnable KGEs. DKRL embeds entities by combining word embeddings of entity descriptions with a CNN encoder, followed by the TransE scoring function. Similarly, BLP embeds raw text using BERT (Devlin et al. 2019). In addition to BLP's approach, KEPLER (Wang et al. 2021b) uses a language modeling objective for training. While these simple approaches indeed improve scalability, they have a negative impact on quality.

SimKGC (Wang et al. 2022) was developed in parallel to KGT5. In contrast to the other bi-encoder approaches, it utilizes two pretrained BERT Transformers: one to embed query entities (and relations) based on their mention and description, and one for tail entities. This approach reaches high quality and was tested on larger graphs, such as Wikidata5M. SimKGC's quality can further be improved by improved training strategies and ensembling multiple SimKGC models (Jiang et al. 2023). However, the applied training strategies (generating hard negative samples) lead to increased training cost limiting the scalability of the approach.

In general, the presented composable bi-encoder approaches offer scalable models. However, for efficient inference, they materialize all compositional embeddings to be able to score a query against all possible answers. Inference cost still scales linearly w.r.t. the number of entities. This reduces the scalability benefits. Additionally, bi-encoder approaches offer limited versatility, similar to KGEs.

Cross-encoder. KG-BERT (Yao et al. 2019) utilizes BERT as a cross-encoder. To retrieve the plausibility of a single triple, KG-BERT encodes a sequence containing subject description, relation mention, and object description with BERT, and feeds the `[CLS]` token into a classification head. This approach holds potential for versatility as it is applicable to downstream NLP tasks. However, the underlying cross-encoder

leads to scalability issues for link prediction, as the query needs to be encoded repeatedly together with all possible answers.

Encoder-sampling. MLMLM (Clouatre et al. 2021) encodes the query with a RoBERTa-based model and uses `[MASK]` tokens to generate predictions. In contrast to the sampling approach applied for KGT5's decoder, MLMLM samples directly from the encoder filling the masked slot. While the underlying idea of generating answer entities is close to KGT5, MLMLM performs significantly worse than atomic KGE models on link prediction on large KGs and is yet to be applied to downstream text-based tasks.

5.2.2 Integrating Neighborhood Information

HittER (Chen et al. 2021) inspired the neighborhood integration of KGT5-context. It is an atomic bi-encoder approach that utilizes a Transformer to encode the query entity together with the query relation. Note that as for traditional KGE models, entities and relations are represented by a unique embedding. To integrate context information, each neighboring connection (entity and relation) is encoded separately via the Transformer. An additional stacked Transformer takes the `[CLS]` tokens from query and neighbor Transformers, and contextualizes the query in its neighborhood. Tail entities are encoded separately and scored together with the contextualized query representation using a classification head. While this approach achieves high quality on small benchmarks, it does not scale to large-scale graphs. Further, the stacked bi-encoder approach limits versatility.

SimKGC (Wang et al. 2022) can similarly integrate neighborhood information. When entity descriptions are limited, Wang et al. (2022) integrate context information by appending mentions of neighboring entities. However, due to a large memory footprint, the integrated context is small, ignores connecting relations, and is only applied on small datasets.

With a simple encoder-decoder architecture and posing link prediction as a seq2seq task, KGT5 fulfills all desiderata. A simple hybrid extension integrating the neighborhood allows for considerable quality improvements. Further, independent follow-up work performing a comparable study, confirms these improvements (Chepurova et al. 2023).

5.3 The KGT5 Model

For a direct integration of textual information, we pose knowledge graph link prediction as a sequence-to-sequence task. We then train a simple encoder-decoder Transformer—that has the same architecture as T5 (Raffel et al. 2020) but without the pretrained weights—on this task. This method, which we call KGT5, results in a scalable, text-based KG link prediction model with vastly fewer parameters than conventional KGE models for large KGs. This approach also confers simplicity and versatility to the model, whereby it can be easily adapted to question answering over KGs on any dataset regardless of question complexity. For more insights into versatility and the application to question answering see Saxena et al. (2022).

Posing KG link prediction as a seq2seq task requires textual representations of entities and relations, and a verbalization scheme to convert link prediction queries to textual queries; these are detailed in Sec. 5.3.1. The link prediction training procedure is explained in Sec. 5.3.2 and inference in Sec. 5.3.3 and 5.3.4.

5.3.1 Textual Representations & Verbalization

Text mapping. We require a one-to-one mapping between an entity/relation and its textual representation, and the mapped textual representation needs to hold useful information describing the entity/relation. For Wikidata-based KGs, we use canonical mentions of entities and relations as their textual representation, followed by a disambiguation scheme that appends unique IDs to duplicate names.

Verbalization. We convert $(s, p, ?)$ query answering to a sequence-to-sequence task by *verbalizing* the query $(s, p, ?)$ to a textual representation. This is similar to the verbalization performed by Petroni et al. (2019), except there is no relation-specific template. For example, given a query *(barack obama, born in, ?)*, we first obtain the textual mentions of the entity and relation and then verbalize it as

"`predict tail: barack obama | born in`".

This sequence is input to the model, and the output sequence is expected to be the answer to this query, "`united states`", which is the unique mention of entity *United States*. When entity descriptions are available, we include "`description: <description of query entity>`" right after the query.

5.3.2 Training KGT5

To train KGT5, we need a set of (input, output)-sequences. For each triple (s, p, o) in the training graph, we verbalize the queries $(s, p, ?)$ and $(?, p, o)$ according to

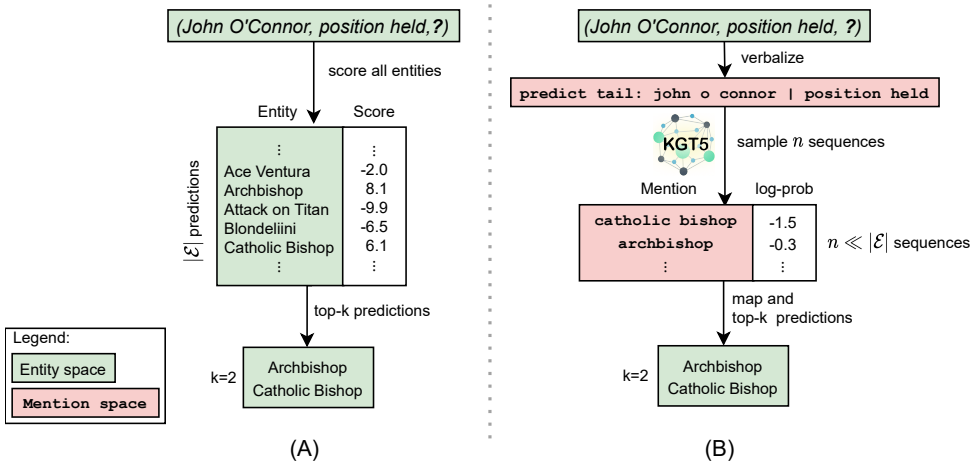


Figure 5.2: Inference pipeline of (A) conventional KGE models versus (B) KGT5 on the link prediction task. Given a query $(s, p, ?)$, we first verbalize it to a textual representation and then input it to the model. A fixed number of sequences are sampled from the model decoder and then mapped back to their entity IDs. This is in contrast to conventional KGEs, where each entity in the KG must be scored. Please see Sec. 5.3.3 for more details.

Sec. 5.3.1 to obtain two input sequences. The corresponding output sequences are the text mentions of o and s respectively. KGT5 is trained with teacher forcing (Williams and Zipser 1989) and cross entropy loss.

One thing to note is that, unlike standard KGE models, we train *without explicit negative sampling*. At each step of decoding, the model produces a probability distribution over possible next tokens. While training, this distribution is penalized for being different from the 'true' distribution (i.e., a probability of 1 for the true next token, 0 for all other tokens) using cross entropy loss. Hence, this training procedure is most similar to the 1vsAll (c.f. Sec. 2.4.1) + cross entropy loss used in Dettmers et al. (2018) and Ruffinelli et al. (2020), except instead of scoring the true entity against all other entities, we are scoring the true token against all other tokens at each step, and the process is repeated as many times as the length of the tokenized true entity. This avoids the need for many negatives and is independent of the number of entities.

5.3.3 Link Prediction Inference

In conventional KGE models, we answer a query $(s, p, ?)$ by finding the score $f(s, p, o)\ \forall o \in \mathcal{E}$, where f is the model-specific scoring function. The entities o are then ranked according to the scores.

In our approach, given a query $(s, p, ?)$, we first verbalize it (Sec. 5.3.1) before feeding it to KGT5. We then *sample* a fixed number of sequences from the decoder,

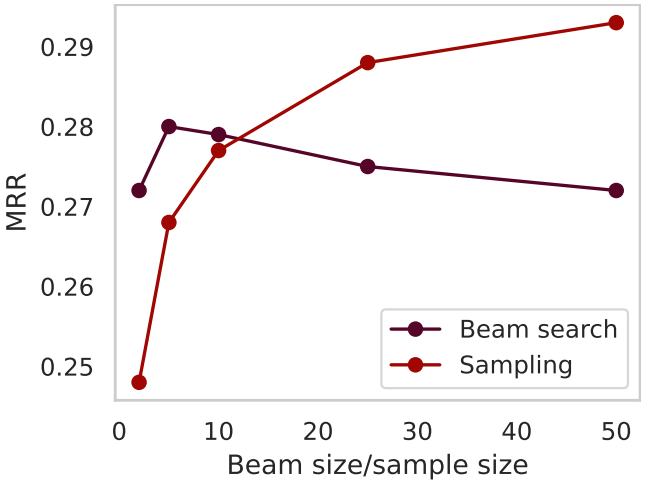


Figure 5.3: Link prediction performance on Wikidata5M. Increasing the sample size steadily increases MRR for the sampling strategy; the opposite effect is seen with beam size ≥ 5 and beam search (also c.f. Yang et al. (2018)).

which are then mapped to their entity IDs.[b] By using such a generative model, we can approximate top-m model predictions without having to score all entities in the KG, as is done by conventional KGE models. However, while in this setting the inference cost is independent of the number of entities, the cost scales quadratically with the sequence length s of the generated text, as well as the depth l of the underlying Transformer. Hence, the generation cost scales by $O(mls^2)$ with $s \ll m \ll |\mathcal{E}|$ and $l \leq 12$ in our setting. For each decoded entity, we assign a score equal to the (log) probability of decoding its sequence. This gives us a set of (entity, score)-pairs. To calculate the final ranking metrics comparable to traditional KGE models, we assign a score of $-\infty$ for all entities not encountered during the sampling procedure. A comparison of the inference strategy of conventional KGE models and KGT5 is shown in Fig. 5.2.

5.3.4 Sampling Strategy for Link Prediction

Multiple strategies to generate multiple high-probability sequences using a decoder have been proposed. A commonly used one is *beam search* (Graves 2012). Beam search allows to deterministically generate as many sequences as needed. Although, while in theory, wider beam sizes should give improved performance, in practice, it has been observed that for beam sizes > 5, the performance of generative models suffers

[b] The decoded sequence may or may not be an entity mention. We experimented with constrained decoding (Cao et al. 2021) to force the decoder to output only entity mentions; however, we found this unnecessary since the model almost always outputs an entity mention, and increasing the number of samples was enough to solve the issue.

drastically (Yang et al. 2018). We observed the same phenomenon for generative link prediction; the MRR decreased with beam size > 5 (see Fig. 5.3).

But, as each query can have multiple correct answers, a large number of unique generated answers is a necessity for high link prediction performance. Possible improvements for beam search in this regard could be modifying the stopping criterion (Murray and Chiang 2018) or training method (Welleck et al. 2019). In this work, however, we opted for a simple sampling approach. In particular, at each step of decoding, we calculate a probability distribution over tokens. We sample a token from this distribution and then autoregressively decode until the 'stop' token. By repeating this sampling procedure multiple times, we can get multiple predictions for the same input sequence. The score for a sequence is the sum of log probabilities for its tokens. For an input sequence $input$ and an entity mention tokenized as $[w_1, w_2, ..., w_T]$, the score for the entity would be

$$\sum_{t=1}^{T} \log(\mathbb{P}(w_t|input, w_1, w_2, ..., w_{t-1})),$$

where $\mathbb{P}$ is the model's output distribution. While this approach sacrifices the determinism offered by beam search, it offers considerably higher performance (see Fig. 5.3).

Note that both approaches could be combined for additional quality improvements, e.g., by generating the first five answer candidates using beam search and sampling all further candidates.

5.4 Expanding KGT5 with Context

To answer a query $(s, p, ?)$, KGT5 has to have facts learned about the query entity s. Under the assumption that the KG will be present during inference, this learning problem can be simplified. Therefore, we extend the verbalization of the query used in KGT5 with information stemming from a graph lookup. We term this approach *KGT5-context*. In particular, we append a textual sequence of the one-hop neighborhood of the query entity s to the verbalized query of KGT5. As a result, the query entity is contextualized, an approach that has been applied successfully before with the HittER model (Chen et al. 2021). While HittER is a non-parametric Transformer model that contextualizes the representation of the query entity with the representations of the neighbors using a stacked transformer, KGT5-context reduces the architecture to a single Transformer working directly on raw-text input. KGT5-context simplifies the prediction problem because additional information that is readily available in the

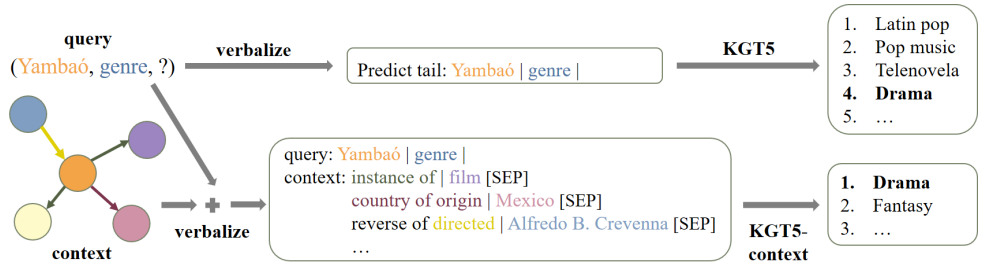


Figure 5.4: Overview of KGT5-context (bottom) and comparison to KGT5 (top); real example from Wikidata5M, best viewed in color. KGT5-context differs from KGT5 in that it appends the neighboring relations and entities of *Yambaó* (a drama movie) to the verbalized query. Both models then apply T5, sample predictions from the decoder, map the samples to entities, and rank by sample logit scores.

KG is provided along with the query. In the example of Fig. 5.4, the contextual information states that *Yambaó* is a Mexican movie. This information is helpful; e.g., it already rules out the top two predictions of KGT5, which incorrectly suggest that *Yambaó* is a piece of music. For a more detailed analysis, see Sec. 5.5.3.

Verbalization details. We obtain mentions of the entities and relations in the query as well as in the one-hop neighborhood of the query entity. We use these mentions to verbalize the query together with the neighborhood as

"`query: <query entity mention> | <query relation mention> | context: <context relation 1 mention> | <context entity 1 mention> <SEP> <context relation 2 mention> | <context entity 2 mention> <SEP> ...`".

When entity descriptions are available, we include "`description: <description of query entity>`" right before the query context. To keep the direction of relations, we prepend the relation mention with "`reverse of`" if the query entity acts as an object, i.e., the relation "points towards" the query entity. A real-world example is given in Fig. 5.4. Inspired by neighborhood sampling in GNNs (Hamilton et al. 2017), we sample up to k (default: $k = 100$) relation-neighbor pairs uniformly, at random, and without replacement.

Dataset	Entities	Relations	|Train|	|Valid|	|Test|
FB15K-237	14 505	237	272 115	17 535	20 466
WNRR	40 559	11	86 835	3 034	3 134
Yago3-10	123 182	37	1 079 040	5 000	5 000
Wikidata5M	4 818 579	828	21 343 681	5 357	5 321
WikiKG90Mv2	91 230 610	1 387	601 062 811	15 000	10 000

Table 5.2: Datasets used in this study.

5.5 Experimental Study

We conducted an experimental study to investigate (i) whether KGT5—i.e. a simple seq2seq Transformer model—can effectively be used for link prediction, (ii) whether an ensemble of the text-based KGT5 model with a KGE model can improve performance, (iii) to what extent integrating context in terms of the entity neighborhood into KGT5 improves link prediction performance, and (iv) for what kind of queries context is helpful. We found that:

1) The text-based KGT5-context improved the state-of-the-art performance on Wikidata5M (Tab. 5.3).
2) KGT5-context was orders of magnitude smaller than the leading models on WikiKG90Mv2 and reached competitive performance (Tab. 5.4).
3) Although standalone KGT5 was good at generalization to unseen facts, it was rather poor at memorizing facts. This problem could be alleviated, if needed, by either integrating context information (using KGT5-context) or using an ensemble of KGT5 and conventional KGE models (Tab. 5.7).
4) Neighborhood information was especially useful if it resided on the shortest path to the answer (Tab. 5.9).

5.5.1 Experimental Setup

We made all source code and configurations publically available at `https://github.com/uma-pi1/kgt5-context`.

Datasets. We evaluate KGT5 and KGT5-context on two commonly used large-scale link prediction benchmarks: Wikidata5M (Wang et al. 2021b), and WikiKG90Mv2 (Hu et al. 2021). Additionally, we considered the small benchmarks FB15k-237 (Toutanova and Chen 2015), WNRR (Dettmers et al. 2018), and Yago3-10 (Dettmers et al. 2018). We use entity and relation mentions provided

with the datasets.[c,d] For Wikidata5M, we also consider the usefulness of entity descriptions, which are provided with the dataset and have been used in some prior studies (Wang et al. 2022; Jiang et al. 2023). Note that we do not use these descriptions by default, and clearly mark throughout when they have been used. In contrast to Wikidata5M, WikiKG90M is only evaluated on tail prediction, i.e.,$(s, p, ?)$ queries. Dataset statistics are summarized in Tab. 5.2. For more details about the datasets see Sec. 2.6.

Metrics. We follow the standard procedure to evaluate model quality for the link prediction task; see Sec. 2.5. Note that for KGT5 and KGT5-context, we do not score all entities during evaluation, but instead sample m times (with $m \ll |\mathcal{E}|$, default $m = 500$) from the decoder and ignore outputs that do not correspond to an existing entity mention.

Settings. For all experiments, we used the same T5 architecture (T5-base for KGT5-context on WikiKG90Mv2, T5-small for everything else) without any pretrained weights. Training from scratch ensures test data is unseen during (pre-)training and avoids leakage. We used the SentencePiece tokenizer pretrained by Raffel et al. (2020). We trained on 8 A100-GPUs with a batch size of 40 (effective batch size of 320) for KGT5 and 32 (effective batch size of 256) for KGT5-context using the AdaFactor optimizer. No dataset-specific hyperparameter optimization was performed. For KGT5-context, we sampled up to 100 neighbors per query entity or up to an input sequence length of 512 tokens. For inference, we obtained 500 samples from the decoder.

Models. On Wikidata5M, we compare KGT5(-context) to the KGE atomic models (only graph structure used) ComplEx (Trouillon et al. 2016), RotatE (Sun et al. 2019), and SimplE (Kazemi and Poole 2018), as well as to the text-based models DKRL (Xie et al. 2016), KEPLER (Wang et al. 2021b), MLMLM (Clouatre et al. 2021), and to the parallel work SimKGC (Wang et al. 2022) and its extension utilizing hard negatives (Jiang et al. 2023). On WikiKG90M, we compare to the models presented on the official leaderboard.[c] Those include atomic KGE models, KGE models combined with text-based MPNet embeddings, as well as ensembles consisting of up to 85 KGE models combined with text-based embeddings. Note that ensemble approaches were proposed after KGT5 but before KGT5-context. For small benchmark KGs, we

[c] We directly used mentions of entities and relations for WikiKG90Mv2, instead of the textual embeddings used by other models. For this reason, the benchmark authors (Hu et al. 2021) did not provide us with scores on the hidden test set. Mentions are provided with the dataset.

[d] For small-scale datasets, mentions and descriptions are provided by (Yao et al. 2019).

Model	MRR	Hits@1	Hits@3	Hits@10	Params	Ent. emb.	1-hop	Pre-trained
RotatE[†]	0.290	0.234	0.322	0.390	2 400M			
SimplE[†]	0.296	0.252	0.317	0.377	2 400M			
ComplEx[††]	0.308	0.255	-	0.398	614M			
SimKGC[e]	0.212	0.182	0.223	0.266	220M	✓		✓
KGT5 (ours)	0.300	0.267	0.318	0.365	60M			
+ ComplEx	0.336	0.286	0.362	0.426	674M			
KGT5-context (ours)	0.378	0.350	0.396	0.427	60M		✓	
DKRL[‡]	0.160	0.120	0.181	0.229	20M	✓		✓
KEPLER[‡]	0.210	0.173	0.224	0.277	125M	✓		✓
MLMLM[‡‡]	0.223	0.201	0.232	0.264	355M	✓		✓
SimKGC + Desc.[$]	0.358	0.313	0.376	0.441	220M	✓		✓
+ Hard Negative Ensemble[$$]	0.420	0.381	0.435	**0.490**	1 100M	✓		✓
KGT5 + Desc. (ours)	0.381	0.357	0.397	0.422	60M			
KGT5-context + Desc. (ours)	**0.426**	**0.406**	**0.440**	0.460	60M		✓	

Table 5.3: Link prediction results on Wikidata5M, test split. The first group does not make use of textual information, the second group uses mention names, and the third group additionally entity descriptions. Best per group underlined, best overall bold. *Ent. emb.* denotes, whether entity embeddings need to be precalculated for inference. *1-hop* denotes, whether the 1-hop neighborhood needs to be retrieved for inference. Marked results are from † Zhu et al. (2019), †† Ch. 3, ‡ Wang et al. (2021b), ‡‡ Clouatre et al. (2021), $ Wang et al. (2022), $$ Jiang et al. (2023).

compared with the currently best-performing link prediction model NBFNet (Zhu et al. 2021).

5.5.2 Link Prediction Performance

Link prediction performance on Wikidata5M is shown in Tab. 5.3. Generally, we found that textual information was highly beneficial. Using mentions only, KGT5 performed on par with the large KGE models and outperformed its bi-encoder counterpart SimKGC by 9pp.[e] Integrating context information into KGT5 improved performance considerably. KGT5-context was the only model that improved upon KGE models (which do not use textual information) when only mention information was available. Entity descriptions provided further improvements; they hold valuable information for this benchmark. With these descriptions, all three approaches—KGT5, KGT5-context, as well as SimKGC—outperformed traditional KGE models by a large margin (12pp in terms of MRR for KGT5-context), with a model size reduction of 90–98% for KGT5(-context) and 80–95% for SimKGC. Both, with and without descriptions, the seq2seq approach KGT5 and especially KGT5-context outperformed the bi-encoder

[e] SimKGC uses descriptions by default. For this ablation, we trained the model with the same hyperparameters but without descriptions.

	Model	Release Date	Test MRR	Valid MRR	Params
Atomic	ComplEx	Sep. 2022	0.141	0.182	18.2B
	TransE	Oct. 2021	0.082	0.110	18.2B
+ Text embedding	ComplEx-Concat	Oct. 2021	0.176	0.205	18.2B
	TransE-Concat	Oct. 2021	0.176	0.206	18.2B
+ Ensemble	PIE-RM	Oct. 2022	0.212	0.254	18.2B[g]
	DGLKE + Rule Mining	Nov. 2022	0.249	0.292	18.2B[g]
	BESS	Dec. 2022	**0.254**	0.292	23.3B[g]
Raw text	KGT5, T5 small (ours)[c]	Mar. 2022	-	0.221	60M
	KGT5-context, T5 base (ours)[c]	May 2023	-	**0.301**	220M

Table 5.4: Link prediction results on WikiKG90Mv2. Baseline numbers are from the official leaderboard of OGB-LSC (Hu et al. 2021).

approach SimKGC and set a new state-of-the-art.[f] Only utilizing expensive negative sampling approaches and an ensemble of multiple SimKGC models, the approach reached performance close to KGT5-context. Previous text-based approaches (DKRL, KEPLER, MLMLM), did not reach results close to the state-of-the-art.

The results on the much larger WikiKG90Mv2 are shown in Tab. 5.4.[c] Here, KGT5 outperformed KGE models and their combination with text embeddings. Note that these text embeddings are based on the same textual data used with KGT5(-context). Only large (300M-45B× larger) and expensive ensembles further improved upon KGT5.[g] But, with context-integration and a slightly larger Transformer architecture (T5-base vs. T5-small), KGT5-context further improved validation MRR by almost 1pp over ensemble approaches. Even with a larger Transformer architecture, KGT5-context is multiple orders of magnitude smaller than the currently best-performing models.

Tab. 5.5 shows link prediction performance on KGs with $\leq 150K$ entities. Here KGT5 and KGT5-context sometimes fall behind the baselines; Transformer models are known to struggle when data is scarce, and this could be the reason for poor performance on these small datasets. Utilizing a pretrained Transformer might help as shown by Wang et al. (2022).

[f] For SimKGC, we used the same hyperparameters as reported for the Wikidata5M dataset in the original paper. However, training for more epochs might further improve SimKGC's performance.

[g] The parameter count in Tab. 5.4 corresponds to the size of the largest model in an ensemble, not the overall model size. For example, BESS (Cattaneo et al. 2022) consists of 85 models, and the complete ensemble has 2.6T parameters; the KGT5-context model is 5 orders of magnitude smaller.

Model	WNRR			FB15K-237			YAGO3-10		
	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10
ComplEx	0.475	0.438	0.547	0.348	0.253	0.536	**0.551**	**0.476**	**0.682**
NBFNet (Zhu et al. 2021)	**0.551**	**0.497**	**0.666**	**0.415**	**0.321**	**0.599**	-	-	-
KGT5	0.508	0.487	0.544	0.276	0.210	0.414	0.426	0.368	0.528
KGT5-context	0.502	0.470	0.569	0.298	0.233	0.439	0.410	0.367	0.484
KGT5-ComplEx Ensemble	0.542	**0.507**	0.607	0.343	0.252	0.377	**0.552**	**0.481**	0.680

Table 5.5: Link prediction results on small KGs ($\leq$ 150k entities). KGT5 is generally worse than both NBFNet and ComplEx on FB15k-237 and YAGO3-10 datasets. Performance on WNRR is somewhat better; however, a part of this could be due to the use of entity definitions (see Sec. 5.6.1).

5.5.3 Context Analysis

To study the impact of context information, we investigated the following questions:

(i) How much context can be used?,

(ii) How much context is useful?,

(iii) In which cases is context useful?,

(iv) What kind of context is useful?,

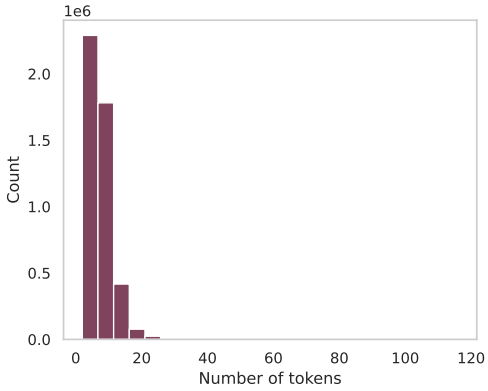
(v) Does context have a similar effect in a bi-encoder architecture?

How much context can be used?

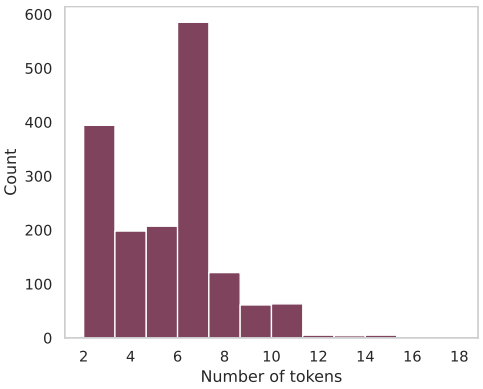
We provide an overview of the lengths of tokenized entities and relations in Fig. 5.5a and Fig. 5.5b respectively. Further, we tokenized each entity together in its one-hop neighborhood; results are summarized in Fig. 5.5c. For 99% of entities, the query including full context information does fit into the default input length of 512 tokens. This does still hold when taking entity descriptions into account (Fig. 5.5d). Larger input lengths are not necessary.

How much context is useful?

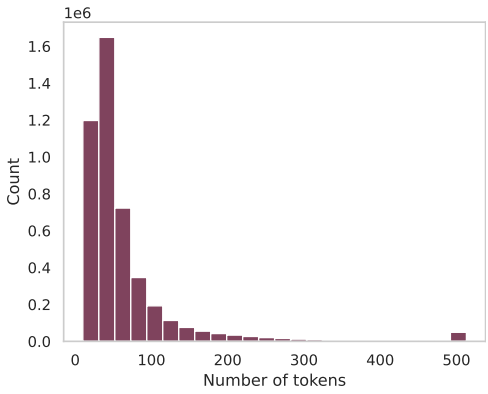
To investigate the effect of context size on quality, we trained multiple KGT5-context models for a single epoch with increasing context size during training, as well as during evaluation. Results are summarized in Tab. 5.6. Performance improved with increasing context size up to 50. Improvements stagnate most likely due to 99% of entities having a degree $\leq$ 50. The results further indicate that training cost could be reduced by limiting the context size during training to 10 but keeping a larger size for inference. Variance is negligible in all settings.



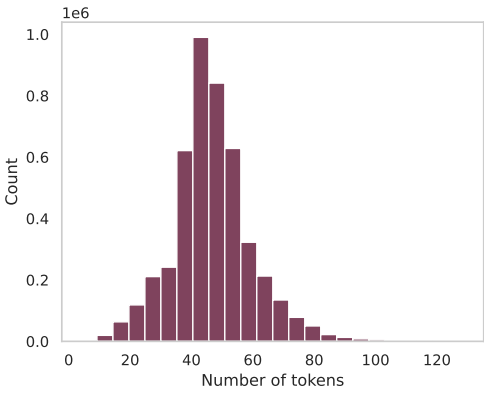
(a) Number of tokens per entity mention in Wikidata5M.



(b) Number of tokens per relation mention in Wikidata5M.



(c) Number of tokens per entity together with its one-hop neighborhood in Wikidata5M. Lengths > 512 fall into bin 513.



(d) Number of tokens per entity description in Wikidata5M. Lengths > 128 fall into bin 129.

Figure 5.5: Length of tokenized entities and relations.

In which cases is context useful?

To investigate in which cases context information was beneficial, we empirically analyzed LP performance w.r.t. (i) query frequency and (ii) the degree of the query entity.

Query frequency. The *frequency* of a test query $(s, p, ?)$ is the number of answers to the query already available in the training data. For example, queries for N:1 relations have frequency 0, whereas queries for 1:N relations can have a large frequency for high-degree query entities. We bucketized the test queries of Wikidata5M into low, medium, and high frequency queries and report average MRR for various models in Tab. 5.7. Generally, high-frequency queries appear harder to answer. These queries have many known true answers already (tying up model capacity); there may be many additional, potentially unrelated answers, and incompleteness of the KG may

Evaluation → Training ↓	1	10	50	100
1	0.280	0.296	0.284	0.284
10	0.269	0.342	0.354	0.354
50	0.259	0.340	0.357	0.357
100	0.258	0.339	0.356	0.356

Table 5.6: Influence of context size on valid-MRR. Rows define the context size during training, and columns during evaluation. Each model was trained for a single epoch on Wikidata5M. Mean over three evaluation runs. Standard deviation is negligible.

Model	0	1-10	>10	All
ComplEx	0.534	0.351	**0.045**	0.296
KGT5	0.624	0.215	0.015	0.300
KGT5-context	**0.738**	**0.415**	0.014	0.378
KGT5 + ComplEx	0.624	0.351	0.045	0.336
KGT5-context + ComplEx	**0.738**	0.351	**0.045**	**0.379**

Table 5.7: Test MRR on Wikidata5M grouped by query frequency during training.

be a concern during evaluation. In contrast, a low-frequency query such as *(Brendan Fraser, instance Of, ?)* has few or no known answers and might be easier to infer, even when the combination of this particular subject and relation was not yet seen during training.

Ensemble with KGE models. In general, the KGT5 model performed reasonably well on queries that did not occur in the training data but was outperformed by a large amount by ComplEx on queries seen multiple times. Hence, both models complemented each other in an ensemble. For this ensemble, we used KGT5 if the query did not have answers in the train KG; otherwise, we used ComplEx. While such an ensemble improves quality over both models separately, it neither achieves the goal of scalability nor versatility. KGT5-context strongly improved performance over ComplEx, KGT5, and the KGT5+Complex ensemble for low- and medium-frequency queries. For this reason, an ensemble between KGT5-context and ComplEx only brought negligible benefits but had substantial drawbacks. Consequently, an ensemble of KGT5-context with a KGE model is not needed and should not be used.

Entity degree. We also investigated the benefit of contextual information w.r.t. to the degree of the query entity (see Fig. 5.6). We found that KGT5-context was beneficial and performed well on query entities with a degree of up to 100. For entities with very large degrees (i.e., nodes with more than 100 or even 1000s of neighbors), ComplEx

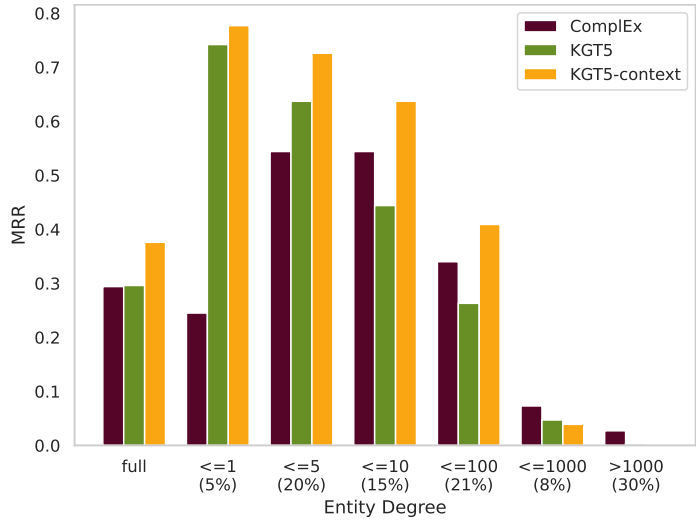


Figure 5.6: MRR grouped by entity degree on Wikidata5M. Group weight is given in brackets.

Inference Setting	Max. context size	MRR	MRR-head	MRR-tail
Default	100	0.3601	0.0743	0.6477
Best context (leakage)	1	0.3830	0.0964	0.6696

Table 5.8: Performance of KGT5-context with the best possible selected context on 500 sampled Wikidata5M validation triples. Context selection was performed exhaustively, knowing the correct answer upfront.

showed benefits. As before, we feel that these performance benefits are negligible considering the increase in model size and decrease in scalability.

What kind of context is useful?

Exhaustive search. To investigate what kind of context useful is, we sampled 500 validation triples and exhaustively ranked the correct answer using a query with each possible context triple separately. Therefore, we ran each query with a context size of 1 as often as the degree of the query entity. The possible performance, when selecting the best possible context is summarized in Tab. 5.8. Note that this approach introduces leakage and cannot be used in production. The correct answer needs to be known upfront for this analysis. In contrast to the results presented in Tab. 5.6, this exhaustive analysis showed that a context size of 1 can be enough or even better than a large size of 100 if selected properly. While this exhaustive search showcased that there may be potential for improvements of KGT5-context, we did not find a selection strategy without leakage performing close to using large context windows.

Pattern	Head	Tail
Contains answer	6.6%	6.4%
Contains query relation	3.6%	6.0%
Lies on shortest path	26.4%	58.8%

Table 5.9: Percentage of cases in which the best-selected context (i) contains the answer entity, (ii) contains the query relation, or (iii) lies on the shortest path to the answer. Analyzed on 500 sampled Wikidata5M validation triples.

Pattern analysis. To find out what context was the best, we analyzed how often (i) the best context entity was the answer entity, (ii) the best context relation was the query relation, and (iii) the best context resides on the shortest path to the answer entity. Results are summarized in Tab. 5.9. For Wikidata5M, the correct answer entity appears in the one-hop neighborhood of the query entity for about 7% of the validation triples. In these cases, the best-selected context was the current connection to the answer entity itself. Utilizing the same relation as context was only beneficial for some relation types. The most beneficial by far, was the context that lies on the shortest to the answer entity. This was the case for almost 60% of tail prediction queries.

Anecdotal results. We manually probed some predictions of KGT5-context and found, in addition to previous insights, the context to be beneficial when the query entity mention only provided limited information about the entity itself. A case of this is shown in Fig. 5.4, a real example. Here, KGT5 was able to capture the geographic region of the real-world entity only based on its mention. Based on this geographic notion, it proposed the music genre *Latin pop* but was unaware that the entity is a movie. This useful information can be obtained directly from the one-hop neighborhood and, indeed, was exploited by KGT5-context. Further, in some cases, even though the answer does not directly appear in the context, it may contain entities strongly hinting at the correct answer. For example, it is easier to predict that an entity has occupation *biochemist*, when the context already contains the information that the entity is a *chemist*. In multiple cases, these situations overlap with the context entity lying on the shortest path to the answer.

To what extent is context useful for a bi-encoder approach?

KGT5-context includes neighborhood information of the query entity. A similar approach can be taken for SimKGC. However, the architecture additionally allows contextualizing the target entity in its neighborhood. To analyze the impact of this integration, we trained SimKGC on Wikidata5M with 10 neighbors for query and

Setting	MRR
SimKGC + desc.	0.358
SimKGC-context + desc.	OOM
SimKGC-DistillBert + desc.	0.361
SimKGC-DistillBert-context + desc.	0.397

Table 5.10: Context information is beneficial for the bi-encoder approach SimKGC. MRR on Wikidata5M.

target entities. Results are summarized in Tab. 5.10. Since SimKGC consists of two separate BERT encoders, its memory overhead is high. The additional neighborhood information exceeds the GPU-VRAM. Therefore, we exchange BERT with the slightly smaller DistillBERT (Sanh 2019). This exchange does not impact performance. The neighborhood contextualization shows similar (but not as strong) performance improvements for SimKGC as it does for KGT5-context.

5.5.4 Impact of Raw Text Information

KGT5-context and SimKGC showed with the integration of detailed textual information strong improvements over link prediction models relying on structural information only (see Tab. 5.3). While text-augmented graphs are prevalent, in many graph settings detailed text information might not be provided. Here, we investigate to what extent the performance gain came from the raw text information and if both model types are useful without textual input using only feature-based and structural information.

We performed a two-step analysis. In the first step, we kept textual information but condensed it to a low-dimensional input embedding. For this analysis, we embedded mentions and descriptions using MPNet (Song et al. 2020) as done for the WikiKG90M-v2 benchmark (Hu et al. 2021). Note, that the answer entity mention for tail prediction was oftentimes directly contained in the entity description of the query entity (31% of tail prediction and 1% of head prediction queries on Wikidata5M valid split). With this first step, we kept the valuable textual information which most likely had a strong influence on final performance. However, without raw text input but the information stored in an embedding, a direct copy of the answer string might not be possible. In the next step, we discarded text information and evaluated performance with structural input features like knowledge graph embeddings. For this analysis, we used ComplEx trained on Wikidata5M.

For SimKGC, we directly replaced the raw text input with feature representations (padded to the default BERT input dimension). Additionally, we contextualized entity

Model	Vocab. learning	Feature input	MRR
ComplEx	-		0.308
SimKGC + desc.	tokenizer	raw text	0.358
SimKGC-DistillBert-context + desc.	tokenizer	raw text	0.397
KGT5-context + desc.	tokenizer	raw text	0.426
SimKGC-context	-	ComplEx	0.280
KGT5-context	PQ	ComplEx	0.268
KGT5-context	FSQ-VAE	ComplEx	0.260
SimKGC-context	-	MPNet	0.355
KGT5-context	PQ	MPNet	0.299
KGT5-context	FSQ-VAE	MPNet	0.291

Table 5.11: Performance of text-based models drops significantly with feature-based input. MRR on Wikidata5M.

representations with the feature representation of its one-hop neighborhood, similar to the contextualization of KGT5-context. For KGT5-context, direct replacement of raw text with feature embeddings is not advisable. For generation, the model should represent each entity by a set of tokens using an underlying vocabulary and represent each token by an embedding. With direct replacement, each entity would be represented by a separate unique token. Here, generation would be performed via sampling in a single step from the complete set of entities. To create a useful vocabulary over the input features, we utilized an adaption of product quantization (PQ) (Jegou et al. 2010), as well as a finite scalar quantized variational autoencoder (FSQ-VAE) (Mentzer et al. 2023). For more details on the vocabulary learning step, see Appendix B.1. Results are summarized in Tab. 5.11.

Both approaches showed a significant drop in performance over their raw-text counterparts. With neighborhood contextualization, the bi-encoder approach could recover the performance of the raw-text approach without contextualization. This contextualization was possible due to smaller memory overhead; each entity (including description) was represented by a single embedding only. However, contextualized SimKGC using a smaller model with raw text input still outperformed the feature-based approach. Since KGT5-context only used the feature input implicitly by first learning a vocabulary over the input, the drop in performance is even more significant.

With structural information only, using ComplEx embeddings, performance dropped further. The models could barely recover the performance of the underlying structural model ComplEx. Therefore, it can be concluded that the performance improvements of both modeling approaches mainly stem from the beneficial use of

raw text information. If it is not provided, the architectures do not bear benefits over previously introduced models.

5.6 Conclusion

We have shown that KG link prediction can be treated as a seq2seq task and tackled successfully with a single encoder-decoder Transformer model. We did this by training a Transformer model with the same architecture as T5 on the link prediction task. This simple but powerful approach, which we call KGT5, performed competitively with the state-of-the-art methods for KG completion on large KGs while using up to 98% fewer parameters. Additionally, KGT5-context extends the KGT5 model by using contextual information of the query entity for prediction. Integrating contextual information simplifies the learning problem and further improves upon the current state-of-the-art.

5.6.1 Limitations

KGT5(-context) relies on the textual mentions of entities and relations (and, optionally, entity descriptions). Therefore, it is only applicable to KGs that provide such information. KGT5-context may be able to handle some entities without textual features when well-described by their neighborhood; we did not investigate this though. To use KGT5-context for prediction, the KG has to be queried to obtain context information, i.e., the one-hop neighborhood of the query entity. KGT5-context thus cannot be used without the underlying KG. The verbalized neighborhood of the query entity leads to long input sequences, which in turn may induce higher memory consumption and higher computational cost during training. Overall, training KGT5-context is typically more expensive than training traditional KGE models, which can be tuned (Ch. 4) and trained efficiently (Ch. 3). For inference, KGT5-context first samples relation-neighbor pairs for contextualization, and then samples possible answers from the decoder. These sampling steps can lead to variance in predictive performance. We found this effect to be negligible on Wikidata5M, but it may be larger on other datasets.

5.6.2 Future Work

GPT for link prediction. As this study showed that a generative approach can be used for link prediction, it opens a door for the direct application of pretrained LLMs like GPT 4 and Gemini. Multiple following studies evaluated (i) whether GPT can be used without further finetuning on link prediction in KGs (Chepurova et al. 2023;

Veseli et al. 2023), and (ii) whether LLMs can store information of KGs (Sun et al. 2023). W.r.t. (i), the performance is still far from state-of-the-art even though GPT has (most likely) seen the full KG during pretraining, and (ii) LLMs further show limitations in representing factual knowledge. Hence, future work could study how to improve the factual correctness of LLMs, as well as improve memorizing capabilities.

Context selection and increased context size. KGT5-context offers a suitable baseline for contextualized link prediction. However, the current context selection is limited to the sampled one-hop neighborhood. A natural direction, for example, is to explore approaches that integrate contextual information in a less naive way than KGT5-context does. Here, an integration of rule-based link prediction approaches like AnyBurl (Meilicke et al. 2019) might be helpful. Next to context selection, an increased context window for the integration of multi-hop neighborhoods might be beneficial. The context window could be increased by utilizing Transformers allowing long input sequences, such as BigBird (Zaheer et al. 2020) or Reformer (Kitaev et al. 2020), or by summarizing the prompt including the neighborhood using so-called "gist-tokens" (Mu et al. 2023).

Improved sampling methods. For text generation, KGT5 relies on random sampling. Future work should investigate if improvements in sampling techniques, such as, nucleus sampling (Holtzman et al. 2019) and unique randomizer (Shi et al. 2020), translate to improvements in seq2seq link prediction. With random sampling, KGT5 oftentimes samples the same sequence multiple times. Considerably improving the variety of sampled sequences, an integration of the unique randomizer would increase the probability that the sequence corresponding to the correct answer entity will be contained in the set of sampled sequences.

Multilingual link prediction. While most research focuses on link prediction in monolingual graphs, recent work proposed to provide KGs to a wider community through multilingualism (Han and Gardent 2023; Singh et al. 2021). A straightforward approach to allow KGT5 to use information stemming from multiple languages would be to exchange the underlying T5 model with mT5 (Xue et al. 2021). In this case, KGT5 should utilize the pretrained multilingual Transformer and finetune for link prediction only.

Chapter 6

Semi-Inductive Link Prediction

"There are things known and there are things unknown, and in between are the doors of perception."

Aldous Huxley, n.d.

In previous chapters, we focused on transductive link prediction. In this setting, all entities are known a priori. In contrast, semi-inductive link prediction in knowledge graphs is the task of predicting facts for new, previously unseen entities based on context information. Although new entities can be integrated by retraining the model from scratch in principle, such an approach is infeasible for large-scale KGs, where retraining is expensive and new entities may arise frequently. In this chapter, we evaluate models' abilities to model emerging entities. To do so, we propose and describe a large-scale benchmark to evaluate semi-inductive link prediction models. The benchmark is based on and extends Wikidata5M: It provides transductive, k-shot, and 0-shot link prediction tasks, each varying the available information from (i) only KG structure, to (ii) including textual mentions, and (iii) detailed descriptions of the entities. Using this benchmark, we report on a small study of recent approaches and found that semi-inductive link prediction performance is far from transductive performance on long-tail entities throughout all experiments. The benchmark provides a test bed for further research into integrating context and textual information in semi-inductive link prediction models.

This chapter is based on Kochsiek and Gemulla (2023).

6.1 Introduction

Previous chapters focused on transductive (TD) link prediction with all entities known a priori. New links are only predicted between entities that were part of the training data. However, in many settings, new entities emerge frequently. Addressing these emerging entities, link prediction can be semi-inductive (SI; some entities known a priori), as well as inductive (no entities known a priori). In this chapter, we concentrate on the semi-inductive setting in its interplay with transductive link prediction.

Semi-inductive link prediction focuses on modeling entities that are unknown or unseen during inference, such as out-of-KG entities (not part or not yet part of the KG) or newly created entities, e.g., a new user, product, or event. Such previously unknown entities can be handled by retraining in principle. For large-scale KGs, however, retraining is inherently expensive, and new entities may arise frequently. Therefore, the goal of semi-inductive link prediction is to avoid retraining and perform link prediction directly, i.e., to generalize beyond the entities seen during training.

To perform link prediction for unseen entities, context information about these entities is needed. The amount and form of context information varies widely and may take the form of facts and/or textual information, such as an entity mention and/or its description. For example, a new user in a social network may provide a name, basic facts such as gender or country of origin, and perhaps a textual self-description.

In this chapter, we introduce the *Wikidata5M-SI* benchmark for semi-inductive link prediction to evaluate current models' abilities to model such emerging entities. Our benchmark is based on the popular Wikidata5M (Wang et al. 2021b) benchmark and has four major design goals:

(G1) **Long-tail entities.** It ensures that unseen entities are long-tail entities since popular entities (such as, say, *Germany*) and/or types and taxons (such as human and organization) are unlikely to emerge after the initial construction of the graph.

(G2) **Varying amounts of contextual facts.** It allows to evaluate each model with varying amounts of contextual facts (0-shot, few-shot, transductive), i.e., to explore individual models across a range of tasks.

(G3) **Controlled amount of textual information.** It provides a controlled amount of textual information (none, mention, full description), where each setting demands different modeling capabilities.

(G4) **Large-scale.** The benchmark is large-scale so that retraining is not a suitable approach.

Base	Dataset	G1	G2	G3	G4
WN11	WN11-OOKB (Hamaguchi et al. 2017)				
WN18RR	oWN18RR (Albooyeh et al. 2020) WN18RR-dynamic (Daza et al. 2021)				
FB15K-237	FB15K-237-OWE (Shah et al. 2019) oFB15K-237 (Albooyeh et al. 2020) FB15K-237-dynamic (Daza et al. 2021)	✓			
FB15K	FB20K (Xie et al. 2016) FB15K (Wang et al. 2019)	✓			
Yago3-10	oYago3-10 (Galkin et al. 2021)				
DBPedia	DBPedia50K (Shi and Weninger 2018) DBPedia500K (Shi and Weninger 2018)	✓ ✓			 ✓
Wikidata5M	Wikidata5M-SI (ours)	✓	✓	✓	✓

Table 6.1: Overview of existing SI link prediction benchmarks.

All prior semi-inductive link prediction benchmarks, which we'll discuss in Sec. 6.2, violate at least one of these criteria. Using our introduced benchmark, we report on a small experimental study with recent link prediction approaches. In general, we found that

1) SI performance was far behind TD performance in all experiments for long-tail entities,
2) there was generally a trade-off between TD and SI performance,
3) textual information was highly valuable,
4) proper integration of context and textual information needs further exploration, and
5) facts involving less common relations provided more useful context.

Our benchmark provides directions and a test bed for further research into SI link prediction.

6.2 Related Work

Semi-Inductive benchmarks. Multiple SI link prediction datasets have been proposed in the literature. For a summary of the proposed benchmarks w.r.t. the goals introduced in Sec. 6.1, see Tab. 6.1. The benchmarks WN18RR-dynamic, FB15K-237-dynamic (Daza et al. 2021), oWN18RR, oFB15K-237 (Albooyeh et al. 2020), and oYago3-10 (Galkin et al. 2021) are obtained by first merging the splits of smaller transductive link prediction datasets and subsequently sampling unseen entities

		Transductive		Semi-inductive	
	Train	Valid	Test	Valid	Test
Triples	20 600 187	4 983	4 977	5 500	5 500
Entities	4 593 103	7 768	7 760	3 722	3 793
Entities unseen	-	0	0	500	500
Relations	822	217	211	126	115

Table 6.2: Statistics of the Wikidata5M-SI splits.

uniformly to construct validation and test splits. These benchmarks do not satisfy goals G1–G4. With DBPedia50k and DBPedia 500k Shi and Weninger (2018), and with FB15K-237-OWE Shah et al. (2019) follow a similar approach but focus on only 0-shot evaluation based on textual features only. For FB20k, Xie et al. (2016) select entities from Freebase with connection to entities in FB15K (Bordes et al. 2013), also focussing on 0-shot evaluation using rich textual descriptions. These approaches do not satisfy G2 and G3. Finally, Wang et al. (2019) and Hamaguchi et al. (2017) uniformly sample test triples and mark occurring entities as unseen. These approaches do not focus on long-tail entities (and, in fact, the accumulated context of unseen entities may be larger than the training graph itself) and they do not satisfy G1–G3.

Inductive benchmarks. There are also several fully inductive LP benchmarks (Teru et al. 2020; Wang et al. 2021b) involving KGs. While semi-inductive link prediction aims to connect unseen entities to an existing KG, fully inductive link prediction reasons about a new KG with completely separate entities (but shared relations). Teru et al. (2020) create inductive benchmarks of common small-scale link prediction benchmarks by sampling multiple subgraphs with disjoint entities but common relations. During inductive inference, models have access to a separate inductive inference graph. Next to the transductive Wikidata5M, Wang et al. (2021b) provide a fully inductive split. Here, each new unseen entity comes with textual mentions and descriptions during inference, but without further facts in the form of triples. Galkin et al. (2022) extend this setting by adding an additional inference graph, similar to the benchmarks presented by Teru et al. (2020). In this work, we do not further consider the fully inductive setting but focus with semi-inductive link prediction on the task of connecting emerging entities to an existing KG.

6.3 The Wikidata5M-SI Benchmark

Wikidata5M-SI is based on the popular Wikidata5M (Wang et al. 2021b) benchmark, which is induced by the 5M most common entities of Wikidata. Our benchmark

WikidataID	Mention	All entities	Long-tail entities
Q5	human	39%	61%
Q11424	film	3%	8%
Q484170	commune of France	1%	7%
Q482994	album	3%	1%
Q16521	taxon	9%	1%
Q134556	single	1%	1%
Q747074	commune of Italy	0%	1%
Q2074737	municipality of Spain	0%	1%
Q571	book	1%	1%
Q7889	video game	1%	1%

Table 6.3: Distribution of top 10 entity types over long-tail entities with degrees between 11 and 20 compared to all entities.

contains transductive and semi-inductive valid/test splits; see Tab. 6.2 for an overview. Generally, we aimed to keep Wikidata5M-SI as close as possible to Wikidata5M. We did need to modify the original transductive valid and test splits, however, because they unintentionally contained both seen and unseen entities (i.e., these splits were not fully transductive). We did that by simply removing all triples involving unseen entities.

Unseen entities. To ensure that unseen entities in the semi-inductive splits are from the long tail (G1), we only considered entities of degree 20 or less. To be able to provide sufficient context for few-shot tasks (G2), we further did not consider entities of degree 10 or less. In more detail, we sampled 500 entities of degrees 11–20 (stratified sampling grouped by degree) for each semi-inductive split. All sampled entities, along with their facts, were removed from the train split. Note that these long-tail entities have a different class distribution than entities from the whole KG; see Tab. 6.3 for an overview of the distribution shift for the top 10 entity types. This difference is natural. In particular, high-degree entities in a KG such as Wikidata often refer to types/taxons (e.g., human, organization, ...) as well as popular named entities (e.g., Albert Einstein, Germany, ...). These entities are fundamental to the KG and/or of high interest and have many facts associated with them. For this reason, they do not form suitable candidates for benchmarking unseen or new entities. In addition, removing high-degree entities for the purpose of evaluating SI-LP is likely to distort the KG (e.g., consider removing the type "human" or "Germany"). In contrast, Wikidata5M-SI focuses on entities for which knowledge is not yet abundant: long-tail entities are accompanied by no or few facts (at least initially) and our SI-LP benchmark tests reasoning capabilities with this limited information.

ID	Q18918	
Mention	Sam Witwer	
Description	Samuel Stewart Witwer (born October 20, 1977) is an American actor and musician. He is known for portraying Crashdown in Battlestar Galactica, Davis Bloome in Smallville, Aidan Waite in Being Human, and Ben Lockwood in Supergirl. He voiced the protagonist Galen Marek / Starkiller in Star Wars: The Force Unleashed, the Son in Star Wars: The Clone Wars and Emperor Palpatine in Star Wars Rebels, both of which he has also voiced Darth Maul.	
Context triples	instance of | human	M: ∘ D: ∘
	country of citizenship | United States of America	M: × D: ∘
	occupation | musician	M: × D: ✓
	occupation | actor	M: × D: ✓
	place of birth | Glenview	M: × D: ×
	given name | Samuel	M: ∘ D: ✓
	given name | Sam	M: ✓ D: ∘
	cast member | Battlestar Galactica	M: × D: ✓
	cast member | Being Human - supernatural drama television series	M: × D: ✓
	cast member | Star Wars: The Force Unleashed II	M: × D: ∘
	cast member | The Mist	M: × D: ×

Table 6.4: Example of an entity from the semi-inductive validation set of Wikidata5M-SI. For each triple, we annotated whether the answer is contained in (✓), deducible from (∘), or not contained in (⊗) mention (M) or description (D).

Tasks and metrics. For TD tasks, we follow the standard protocol of Wikidata5M. To construct SI tasks, we include 11 of the original facts of each unseen entity into its SI split; each split thus contains 5,500 triples. This enables up to 10-shot SI tasks (1 fact to test, up to 10 facts for context). For entities of degree larger than 11, we select the 11 facts with the most frequent relations; see Tab. 6.4 for an example. The rationale is that more common relations (such as *instanceOf* or *country*) may be considered more likely to be provided for unseen entities than rare ones (such as *militaryBranch* or *publisher*). We then construct a single k-shot task for each triple (s, p, o) in the SI split as follows. When, say, s is the unseen entity, we consider the LP task $(s, p, ?)$ and provide k additional facts of form (s, p', o') as context. Context facts are selected by frequency as above, but we also explored random and infrequent-relation context in our study. Models are asked to provide a ranking of predicted answers, and we determine the filtered mean reciprocal rank (MRR) and Hits@k of the correct answer (o).

Textual information. For each entity, we provide its principal mention and a detailed description (both directly from Wikidata5M); see Tab. 6.4. This allows to differentiate model evaluation with varying amounts of textual information per entity (G3): (A) atomic, i.e., no textual information, (M) mentions only, and (D) detailed textual descriptions as in Ch. 5. This differentiation is especially important in the SI setting,

as detailed text descriptions might not be provided for unseen entities and each setting demands different modeling capabilities. In fact, (A) performs reasoning only using graph structure, whereas (D) also benefits from information extraction to some extent. We discuss this further in Sec. 6.5.

6.4 Semi-Inductive Link Prediction Models

Recall that link prediction models can be grouped into global, local, and hybrid models (c.f. Sec. 2.3). We briefly summarize, how existing link prediction models can be used directly or extended in a semi-inductive setting.

Global Models

Global models operate on the full graph for inference, e.g., GNNs and rule-based approaches. As most models in this category can reason on a (slightly) different graph during inference than training, they can handle emerging entities and perform semi-inductive link prediction. In this study, we evaluate the rule-based approach *AnyBurl* (Meilicke et al. 2019). We originally planned to consider *NodePiece* (Galkin et al. 2021), representing entities by a combination of anchor embeddings, and *NBFNet* (Zhu et al. 2021)[a], a GNN-based LP model; both support SI-LP directly. However, the available implementations did not scale to Wikidata5M-SI (out of memory).[b]

Local & Hybrid Models

In contrast to global models, local models operate on a triple level. For more details see Sec. 2.3. Multiple local models learn representations of the entities in the graph which are in turn used for inference, e.g., KGE models. As such representations do not yet exist for emerging entities, these models cannot reason about queries containing these entities by default. For SI-LP, models must be able to create such representations "on the fly" given the input features of the emerging entities. Common approaches are (i) the integration of neighborhood information using a hybrid extension and/or (ii) representing entities via their textual features. For (i) the emerging entity is represented by aggregated features of its one-hop neighborhood, and for (ii) by an embedding of textual mentions and description coming with the emerging entity. In

[a] For more details on NBFNet, see Sec. 2.3.2.

[b] For NBFNet (Zhu et al. 2021), the large memory footprint is inherent to the model; it is a full-graph GNN and hard to scale. For NodePiece (Galkin et al. 2021), however, the problem mainly lies in the expensive evaluation. All intermediate representations are precomputed, leading to a large memory overhead.

the following, we summarize local and hybrid models and/or their extensions for SI-LP, which we considered in our experimental study.

Graph-only models. To extend KGE models for SI-LP, we follow a *fold-in* approach explored by Jambor et al. (2021), as well as a hybrid extension proposed by Albooyeh et al. (2020). For the fold-in approach, we first represent each entity as the sum of a local embedding (one per entity) and a global bias embedding. For 0-shot, we solely use the global bias for the unseen entity. For k-shot, we perform the actual fold-in and obtain the local embedding for the unseen entity by performing a single training step on the context triples (keeping all other embeddings fixed). For the hybrid extension, we consider *oDistMult-ERAvg* (Albooyeh et al. 2020). Here, we represent unseen entities by aggregating the embeddings of the relations and entities in the context.[c] A more direct approach is taken by the hybrid *HittER* (Chen et al. 2021), which contextualizes the query entity with its neighborhood for TD-LP. The approach can be used for SI-LP directly by using a masking token (akin to the global bias above) for an unseen entity.

Text-based models. To extend KGE models for SI-LP using textual information, we consider the baseline approach of concatenating learnable representations with pretrained text embeddings explored in the WikiKG90M benchmark (Hu et al. 2021); see Sec. C.1 for details on this integration. The remaining approaches are purely textual. In particular, we considered the approaches we evaluated for TD link prediction in Ch. 5; i.e., *SimKGC* and *KGT5*. Both approaches support 0-shot SI-LP when textual information is provided for the query entity. They do not utilize additional context, however, i.e., do not support k-shot SI-LP. But, as the hybrid model *KGT5-context* extends the input of KGT5 by the one-hop neighborhood of the query entity, it consequently supports k-shot LP directly.[d]

6.5 Experimental Study

We evaluated all presented baseline models in the TD and SI setting on the atomic, mentions, and descriptions dataset. Further, we evaluated in detail which context was most useful and what information was conveyed by textual mentions and descriptions.

Setup. Source code, configuration, and the benchmark itself are available at `https://github.com/uma-pi1/wikidata5m-si`. For hyperparameter optimization for

[c] To address the high memory footprint (Galkin et al. 2021) of oDistMult-ERAvg, we extend it with neighborhood sampling.

[d] Similar to KGT5-context, SimKGC can be and is used in a hybrid setting on small KGs. However, the memory footprint of SimKGC in this hybrid setting is too large for Wikidata5M-SI. For more details on this extension, including its effect on quality and memory, see Sec. 5.5.3.

Model	Transductive		Semi-inductive (num. shots)					Pre-trained
	All	Long tail	0	1	3	5	10	
ComplEx + Bias + Fold in	0.308	0.523	0.124	0.151	0.176	0.190	0.206	no
DistMult + ERAvg	0.294	0.512	-	0.171	0.246	0.295	0.333	no
HittER	0.284	0.512	0.019	0.105	0.153	0.179	0.221	no
AnyBurl	0.353	0.569	.[e]	.[e]	.[e]	.[e]	0.407[e]	no
DistMult + ERAvg + Mentions	0.299	0.535	-	0.187	0.235	0.258	0.280	yes
SimKGC (mentions only)	0.212	0.361	0.220	-	-	-	-	yes
KGT5	0.281	0.542	0.310	-	-	-	-	no
KGT5-context	0.374	0.678	0.220	0.217	0.236	0.259	0.311	no
+ Context hiding[f]	0.371	0.676	0.283	0.263	0.258	0.272	0.316	no
DistMult + ERAvg + Descriptions	0.313	0.585	-	0.278	0.281	0.285	0.292	yes
SimKGC + Descriptions	0.353	0.663	0.403	-	-	-	-	yes
KGT5 + Descriptions	0.364	0.728	**0.470**	-	-	-	-	no
KGT5-context + Descriptions	**0.420**	0.777	0.417	0.420	0.416	0.420	0.437	no
+ Context hiding[f]	0.418	0.775	0.449	0.447	0.443	0.440	0.446	no

Table 6.5: Transductive and semi-inductive link prediction results in terms of MRR on the dataset Wikidata5M-SI. The first group presents results on the atomic, the second on the mentions and the third on the descriptions dataset. "Long tail" describes queries with a query entity of degree ≤ 10. Best per TD/SI in bold. Best per group underlined.

ComplEx (Trouillon et al. 2016), DistMult (Yang et al. 2015), and HittER (Chen et al. 2021), we used the multi-fidelity approach GraSH (Ch. 4) implemented in LibKGE (Broscheit et al. 2020) with 64 initial trials and trained for up to 64 epochs. For fold-in, we reused training hyperparameters and trained for a single epoch on the provided context. For text-based approaches, we used the hyperparameters and architectures proposed by the authors for the transductive split of Wikidata5M. We trained on up to 5 A6000-GPUs with 49GB of VRAM.

Main results. Transductive and SI performance in terms of MRR of all models is presented in Tab. 6.5; Hits@k in Tab. C.1-C.3 in the appendix. Note that overall transductive performance was oftentimes below best reported SI performance. This is due to varying degrees of query entities between both settings. Typically, models perform better predicting new relations for an entity (e.g., the birthplace) than predicting additional objects for a known relation (e.g., additional awards won by a person); c.f., Sec. 5.5.3. For a direct comparison between both settings, we additionally report TD performance on long tail query entities; i.e., query entities as entities with degree ≤ 10 (as in the SI setting).

Atomic (graph-only). TD performance on the long tail was considerably higher than SI performance. As no information was provided for unseen entities, 0-shot was not reasonably possible. Without text-based information, context was a necessity. A simple neighborhood aggregation—entity-relation average (ERAvg)—offered the

	Mention	**Description**
Contained	10%	44%
Deducible	7%	10%
Not contained	83%	46%

Table 6.6: Information about a query answer contained in mentions and descriptions. Annotated for 100 sampled triples from 0-shot valid. For an example, see Tab. 6.4.

best integration of context for neural models. The rule-based approach AnyBurl performed best in the TD as well as the SI setting.[e]

Mentions. Integrating mentions did not improve performance on its own, as provided text information was still limited. However, additionally providing context information during inference (KGT5-context) simplified the learning problem and improved TD performance significantly. But for 0-shot, the limited text information provided with mentions allowed for reasonable performance. To analyze what information is conveyed for 0-shot, we annotated 100 valid triples; see Tab. 6.6. In 10% of cases, the answer was already contained in the mention, and it was deducible in at least 7%. This enabled basic reasoning without any further information. In contrast to the TD setting, KGT5 outperformed its context extension. KGT5-context was reliant on context which was lacking especially during 0-shot. This showed a trade-off between best performance in the SI and TD setting among the text-based models. This trade-off could be mitigated by applying (full and partial) context hiding. With such adapted training, KGT5-context reached a middle ground improving upon its SI performance with only minor losses in the TD setting.[f] However, even with full context (10-shot), performance was still only on par with KGT5. Therefore, context information did not bring any further benefits when text was provided.[g] Note that context information could also be integrated into SimKGC, and might be more beneficial with its underlying bi-encoder architecture, which allows incorporating context for query and target entity. However, due to the memory overhead of this approach, we did not investigate this approach further. For more details on the effects of such an integration on transductive performance, see Sec. 5.5.3. Overall, mentions were

[e] Note that due to a limiting implementation, we allowed AnyBurl access to the contextual facts of other queries during inference. This additional information might slightly inflate the presented SI performance. Given this limitation, we only evaluated AnyBurl in the 10-shot setting. However, this model can perform reasoning given any number of shots.

[f] In 25%/25%/50% of cases, we hid the full context/sampled between 1-10 neighbors/used the full context, respectively.

[g] For more details on the context impact on SI performance of KGT5-context, one could extend the ablation of context hiding and train a model per context size. This would give an indication of the upper SI performance limit of KGT5-context for a specific k-shot setting.

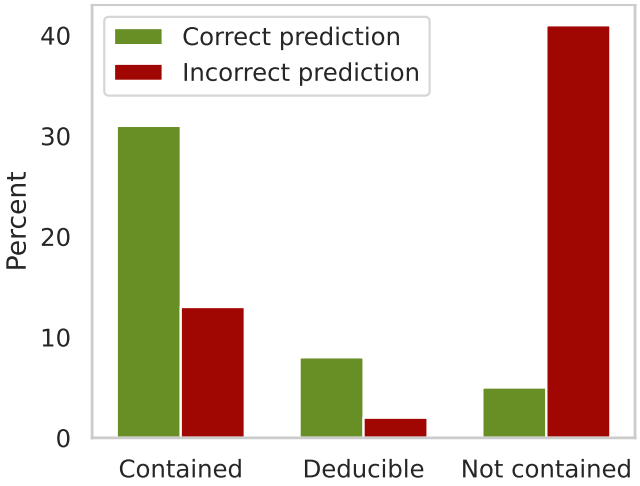


Figure 6.1: Number of correct (rank=1) and incorrect predictions by KGT5+descriptions on annotated examples per annotation label.

Context selection	1	3	5
Most common	0.217	0.236	0.259
Least common	**0.253**	**0.273**	**0.290**
Random	0.237	0.260	0.281

Table 6.7: Influence of context selection. Semi-inductive test MRR of KGT5-context.

mainly beneficial in the TD setting. The rule-based approach AnyBurl outperformed all text-based models relying on mentions for SI-LP.

Descriptions. Further, integrating descriptions improved performance for both settings, TD and SI, considerably; see Tab. 6.5. Similar to the mentions-only setting, KGT5-context performed best in TD and KGT5 in the SI setting. Applying the same trade-off with context-hiding reached a similar middle ground between TD and SI performance.

Descriptions were very detailed and partially contained the correct answer as well as the same information as contained in context triples; see Tab. 6.6. Therefore, performance did not further improve with context size. In such cases, models mainly benefit from information extraction capabilities. To judge how much information extraction helps, we grouped the performance of KGT5+description in the 0-shot setting on validation data into the groups *contained*, *deducible*, and *not contained* in the description; see Fig. 6.1. When contained, the correct answer was extracted in $\approx 70\%$ of cases.

Context selection. We selected the most common relations as context triples so far, as this may be a more realistic setting. To investigate the effect of this selection approach, we compared the default selection of choosing *most common* relations to

least common and *random*. Results for KGT5-context are shown in Tab. 6.7; for all other models in Tab. C.4 in the appendix. We found that the less common the relations of the provided context, the better the SI performance. More common context relations often described high-level concepts, while less common provided further detail; see the example in Tab. 6.4. While the more common context may be more readily available, less common context was more helpful in describing a new entity.

6.6 Conclusion

We proposed the new Wikidata5M-SI large-scale benchmark for semi-inductive link prediction. The benchmark focuses on unseen entities from the long tail and allows evaluating models with varying and controlled amounts of factual and textual context information. In our experimental evaluation, we found that semi-inductive link prediction performance fell behind transductive performance for long-tail entities in general, and that detailed textual information was often more valuable than factual context information. Moreover, current models did not integrate these two types of information adequately, suggesting a direction for future research.

6.6.1 Limitations

This study was performed on Wikidata5M-SI, i.e., a subset of a single knowledge graph. Model performance and insights may vary if graph structure and/or availability and usefulness of mentions and description is different. In particular, the entity descriptions provided with Wikidata5M-SI partly contained information relevant to link prediction so that models benefited from information extraction capabilities. Due to a limiting implementation, we allowed AnyBurl access to the contextual facts of other queries during inference. This additional information might slightly inflate the presented SI performance. The effect of this should be analyzed in detail and implementations made more flexible.

6.6.2 Future Work

Knowledge editing. Connecting emerging entities to the graph extends the overall knowledge base. With this new knowledge, the learned information about the now connected entities might need to be updated next to processing the emerging entities themselves. This update step could be performed by so-called knowledge editing. In general, knowledge editing addresses the need to change any facts in the graph that might have changed in the real world and needs to be updated in the knowledge base.

As for semi-inductive link prediction, these changes are, in the best case, incorporated without model retraining. This task of knowledge editing in language models was first introduced with the KnowledeEditor presented by De Cao et al. (2021). The KnowledgeEditor learns to predict needed weight updates for knowledge editing without affecting the rest of the knowledge. This post-hoc approach could be applied in language model based link prediction, such as SimKGC and KGT5(-context). In contrast, Hewitt et al. (2023) propose to directly adapt the language model architecture to allow for straightforward editing and interpretation. They represent tokens by an aggregation of so-called "sense vectors". Editing the sense vectors of a target word (replacing the association of one word with the association of another) allows for editing relationships between entities. Such a language model could exchange the underlying T5 Transformer utilized in KGT5(-context).

Improved integration of textual information. The integration of detailed textual information considerably improved model performance in the TD, as well as in the SI setting. However, some models relying on graph structure only were able to perform close to models focusing on textual information. Hence, an improved integration of text information into these structure-based models could further improve overall performance. Here, a promising candidate is the rule-based approach AnyBurl with the strongest TD and SI performance among graph-only models.

Chapter 7

Conclusions

This thesis has systematically explored the integration of neural models with multi-relational data, focusing on three primary areas: making multi-relational data accessible for efficient reasoning and downstream applications, integrating structural and textual information, and managing the dynamic nature of evolving multi-relational graphs. A promising candidate for the first area was knowledge graph embeddings. However, for effective integration, such models must be scalable and reach high embedding quality; i.e., capture large-scale relational data well.

Hence, we evaluated state-of-the-art parallel training methods for KGE models on large-scale knowledge graphs, finding that current implementations often degrade model quality compared to sequential training. Nonetheless, efficient and effective training was achievable with careful technique selection. Our experiments demonstrated that a simple random baseline for parallel training, combined with state-of-the-art parameter management systems, is both efficient and effective. Utilizing this random baseline, improvements in parameter servers are likely to directly translate to improvements in parallel KGE training. Considering recent developments in parameter servers as well as hardware improvements, such as faster and larger GPUs, KGE models can be trained efficiently and effectively on a single machine with multiple GPUs with little additional implementation overhead.

Next to parallel training methods, selected hyperparameter configurations have a strong impact on the resulting embedding quality. However, an extensive hyperparameter search leads to excessive cost for KGEs on large-scale graphs. To mitigate

this, we explored various low-fidelity approximation techniques for hyperparameter evaluation of KGE models. Based on our findings, we proposed a multi-fidelity hyperparameter optimizer for KGE models based on successive halving. With such an efficient HPO approach, strong resulting quality can be achieved with little budget. This considerably reduces the entry barrier for applying KGEs to large-scale applications.

Many graphs provide textual information together with their structure. Our second research area focused on integrating structural and textual information. In this context, we demonstrated that this integration could be effectively approached by posing link prediction as a sequence-to-sequence task. This framing allowed to use language models for link prediction, leading to state-of-the-art performance with up to 98% fewer parameters. Given relevant textual information, LLMs can effectively support graph-based tasks. Furthermore, future advancements in LLMs are expected to enhance their performance on text-heavy graph-based tasks. Our approach focused on learning graph data directly; however, it remains to be explored to what extent LLMs should instead learn to access, use, and reason about structured data as needed. One first step in this direction is offered by the research area of retrieval augmented generation.

Lastly, we addressed the dynamic nature of evolving multi-relational graphs. To realistically evaluate link prediction models in dynamic settings, we introduced the large-scale, semi-inductive link prediction benchmark Wikidata5M-SI. We observed that semi-inductive link prediction performance generally lags behind transductive performance for long-tail entities and that detailed textual information is often more valuable than factual context information. Moreover, current models did not integrate these two types of information adequately. This finding again suggests that models applied to structured data should balance learning, retrieving, and reasoning to effectively handle dynamic graph data and allow for extending, editing, and deleting existing information.

Future Work

The presented work opens up multiple avenues for future research, which we outline below.

Our study utilized the task of link prediction to evaluate the ability of models to capture and use relational information. While link prediction serves as a proxy for downstream tasks like recommendation systems, some effects explored in this thesis may not directly translate to the integration of relational knowledge in these applic-

ations. For instance, it remains to be evaluated to what extent training techniques and hyperparameter choices influence downstream performance. On a small scale, Ruffinelli and Gemulla (2024) already showed that strong link prediction performance does not always translate directly to strong performance on other graph-based tasks. A promising starting point for such an investigation on large-scale data is the work by El-Kishky et al. (2022), where the authors trained large-scale knowledge graph embedding models and integrated the resulting embeddings as features in various recommendation models. Building on their methodology, future research could assess the impact of different training and tuning techniques in similar pipelines and further explore optimal strategies for integrating relational knowledge into downstream applications.

While embedding approaches such as KGEs can be directly utilized as features in downstream applications, and relational knowledge can even be integrated into Transformer-based models for natural language interactions, these approaches capture the factual knowledge of the knowledge graph in a somewhat imprecise manner. KGs are constructed with factual correctness in mind, but the methods used to capture this knowledge might not ensure the same level of accuracy in downstream applications. Two possible research directions to address this issue are: (i) integrating logical approaches, and (ii) exploring retrieval-augmented methods.

Regarding (i), rule-based approaches have shown promising results in link prediction tasks and enhanced interpretability. However, they lack integration with textual information and may be challenging to incorporate into certain downstream applications. However, combining neural, text-based and rule-based methods could potentially improve both quality and factual correctness.

Concerning (ii), while language models can learn and reason about structured data, it may be beneficial to store knowledge externally and focus on interaction learning. By storing factual knowledge outside the model parameters and retrieving it when necessary, we can ensure that the retrieved knowledge aligns with the factual accuracy provided by the KG. This approach also facilitates the handling of frequently updated knowledge, thereby avoiding the need for expensive retraining. Further exploring memory modules, as discussed by Févry et al. (2020) and Li et al. (2022a), could enable separate knowledge storage, allowing for extensions, deletions, and updates. Additionally, reinforcement learning could aid Transformers in learning when and how to retrieve information from underlying graphs or memory, potentially performing retrieval iteratively.

Additional Material to Chapter 4

		Variant →	Hits@1			Hits@10		
	Dataset	Model ↓	Epoch	Graph	Comb.	Epoch	Graph	Comb.
Small	Yago	ComplEx	**0.460**	0.375	0.455	**0.672**	0.634	0.660
	3-10	RotatE	0.337	0.337	**0.342**	**0.619**	**0.619**	0.607
	(E = 400)	TransE	**0.406**	0.311	**0.406**	**0.661**	0.628	**0.661**
Medium	Wiki-	ComplEx	**0.247**	**0.247**	**0.247**	**0.390**	**0.390**	**0.390**
	data5M	RotatE	**0.187**	0.169	**0.187**	**0.331**	0.326	**0.331**
	(E = 64)	TransE	0.210	0.210	**0.213**	0.358	0.358	**0.363**
Large	Free-	ComplEx	0.486	**0.511**	**0.511**	0.714	**0.726**	**0.726**
	base	RotatE	0.522	**0.578**	**0.578**	0.625	**0.669**	**0.669**
	(E = 10)	TransE	0.078	**0.520**	**0.520**	0.518	**0.614**	**0.614**

Table A.1: Model quality in terms of Hits@1 and Hits@10. Best reduction variant in bold. GraSH with default settings (B = 3, n = 64, η = 4). Model dim = 128.

Appendix B

Additional Material to Chapter 5

B.1 Building a Vocabulary over Features

To enable KGT5(-context) to work with feature input, rather than raw text, we learn a vocabulary over the provided features. Here, we evaluate two approaches; (i) a simplified version of *product quantization*, and (ii) *finite scalar quantization*.

Product quantization. (Jegou et al. 2010) is typically used as an efficient index for MIPS; e.g., in Faiss (Douze et al. 2024). Here, the feature vector is split into n chunks. For each chunk over all inputs (entity + relation), we train kmeans with k centroids. Each centroid is treated as a token of our vocabulary. The resulting vocabulary size is $n \times k$. Each entity is represented by n tokens. However, it is not guaranteed that resulting entity representations are unique. With a resulting vocabulary size of 32K tokens only $3/4$ of entities have a unique representation; see Tab. B.1. Therefore, for a fair evaluation, we treat all entities with the same representations as ties before calculating MRR.

Finite scalar quantization. Recent work shows that for image, video, and sound input data, vector quantization (VQ) (Gray 1984) in combination with variational autoencoders (VAE) (Van Den Oord et al. 2017) allows for strong autoregressive solutions (Esser et al. 2021; Villegas et al. 2022). In this setting, VQ-VAE is utilized to learn a vocabulary over the input data. However, the underlying codebooks for VQ-VAE are hard to train and often underutilized. A drop-in replacement for VQ is finite scalar quantization (FSQ) (Mentzer et al. 2023). Here, each scalar in the input feature vector is independently quantized. Using an FSQ-VAE, we reach a high

Setting	Vocab. size	Unique entity representations
Default	32 128	4 594 485
Product Quantization ($n = 8$, $k = 4000$)	32 000	3 432 648
FSQ-VAE	16 000	4 458 108

Table B.1: Unique entity representations given the trained vocabulary over MPNet feature input embeddings on Wikidata5M.

percentage of independent coded entities with smaller vocabulary sizes compared to product quantization; see Tab. B.1.

ADDITIONAL MATERIAL TO CHAPTER 6

C.1 Integrating Text into KGE Models

To integrate text into traditional KGE models, we follow the baseline models of the WikiKG90M link prediction challenge (Hu et al. 2021). We embed mentions combined with descriptions using MPNet (Song et al. 2020), concatenate the resulting descriptions embedding with the entity embedding, and project it with a linear layer for the final representation of the entity. In combination with oDistMult-ERAvg (Albooyeh et al. 2020), we apply the aggregation of neighboring entities and relations on the entity embedding part only. The resulting aggregation is then concatenated with its description and finally projected.

This approach is closely related to BLP (Daza et al. 2021). The main differences to BLP are:

1) Hu et al. (2021) use MPNet, BLP uses BERT.
2) In combination with DistMult-ERAvg, we concatenate a learnable "structural embedding" to the CLS embedding of the language model, whereas BLP does not.

		Semi-inductive (num. shots)				
Model	Trans.	0	1	3	5	10
Complex + Bias + Fold in	0.260	0.058	0.097	0.118	0.124	0.132
DistMult + ERAvg	0.237	-	0.115	0.151	0.185	0.209
HittER	0.234	0.005	0.076	0.115	0.132	0.153
DistMult + ERAvg + Mentions	0.239	-	0.106	0.142	0.153	0.167
SimKGC (mentions only)	0.182	0.187	-	-	-	-
KGT5	0.249	0.263	-	-	-	-
KGT5-context	0.347	0.184	0.177	0.195	0.218	0.263
+ Context hiding	0.344	0.241	0.215	0.214	0.227	0.268
DistMult + ERAvg + Descriptions	0.252	-	0.152	0.153	0.153	0.161
SimKGC + Descriptions	0.311	0.349	-	-	-	-
KGT5 + Descriptions	0.332	0.430	-	-	-	-
KGT5-context + Descriptions	0.400	0.379	0.382	0.373	0.378	0.393
+ Context hiding	0.399	0.409	0.407	0.400	0.397	0.401

Table C.1: Transductive and semi-inductive link prediction results in terms of H@1 on the dataset Wikidata5M-SI.

		Semi-inductive (num. shots)				
Model	Trans.	0	1	3	5	10
ComplEx + Bias + Fold in	0.337	0.165	0.180	0.202	0.219	0.242
DistMult + ERAvg	0.328	-	0.190	0.292	0.352	0.401
HittER	0.309	0.013	0.109	0.158	0.188	0.242
DistMult + ERAvg + Mentions	0.332	-	0.239	0.289	0.314	0.340
SimKGC (mentions only)	0.223	0.227	-	-	-	-
KGT5	0.296	0.332	-	-	-	-
KGT5-context	0.390	0.236	0.234	0.257	0.278	0.335
+ Context hiding	0.389	0.300	0.283	0.278	0.292	0.342
DistMult + ERAvg + Descriptions	0.344	-	0.368	0.373	0.378	0.380
SimKGC	0.367	0.421	-	-	-	-
KGT5 + Descriptions	0.385	0.490	-	-	-	-
KGT5-context + Descriptions	0.432	0.441	0.443	0.443	0.447	0.463
+ Context hiding	0.433	0.472	0.469	0.469	0.467	0.476

Table C.2: Transductive and semi-inductive link prediction results in terms of H@3 on the dataset Wikidata5M-SI.

		Semi-inductive (num. shots)				
Model	Trans.	0	1	3	5	10
ComplEx + Bias + Fold in	0.387	0.231	0.245	0.282	0.309	0.336
DistMult + ERAvg	0.389	-	0.270	0.409	0.493	0.564
HittER	0.376	0.050	0.157	0.226	0.270	0.359
DistMult + ERAvg + Mentions	0.411	-	0.320	0.392	0.440	0.478
SimKGC (mentions only)	0.266	0.283	-	-	-	-
KGT5	0.344	0.398	-	-	-	-
KGT5-context	0.423	0.293	0.295	0.310	0.336	0.400
+ Context hiding	0.421	0.367	0.354	0.343	0.354	0.408
DistMult + ERAvg + Descriptions	0.425	-	0.465	0.472	0.484	0.491
SimKGC	0.432	0.504	-	-	-	-
KGT5 + Descriptions	0.416	0.544	-	-	-	-
KGT5-context + Descriptions	0.455	0.484	0.489	0.489	0.495	0.516
+ Context hiding	0.454	0.523	0.521	0.517	0.517	0.522

Table C.3: Transductive and semi-inductive link prediction results in terms of H@10 on the dataset Wikidata5M-SI.

Model	Context selection	1	3	5
ComplEx + fold-in	Most common	0.151	0.161	0.168
	Least common	**0.166**	0.185	0.195
	Random	0.164	**0.187**	**0.196**
DistMult + ERAvg	Most common	0.171	0.246	0.295
	Least common	**0.217**	0.299	**0.323**
	Random	0.215	**0.303**	0.318
oDistMult + ERAvg + Mentions	Most common	0.187	0.235	0.258
	Least common	**0.237**	**0.274**	**0.279**
	Random	0.232	0.265	0.272
HittER	Most common	0.105	0.153	0.179
	Least common	**0.151**	**0.195**	**0.216**
	Random	0.136	0.190	0.206
KGT5-context	Most common	0.217	0.236	0.259
	Least common	**0.253**	**0.273**	**0.290**
	Random	0.237	0.260	0.281
KGT5-context + Desc.	Most common	0.420	0.416	0.420
	Least common	**0.423**	0.424	**0.430**
	Random	0.422	**0.430**	**0.430**

Table C.4: Influence of context selection. Semi-inductive test MRR. Best per model in bold.

Bibliography

Ralph Abboud, Ismail Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. 2020. Boxe: A Box Embedding Model for Knowledge Base Completion. In *Advances in Neural Information Processing Systems*, Vol. 33. 9649–9661.

Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. 2019. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. In *2019 IEEE 35th International Conference on Data Engineering*. IEEE, 1250–1261.

Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. 2023. Similarity Search in the Blink of an Eye with Compressed Indices. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3433–3446.

Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi. 2020. Out-of-Sample Representation Learning for Knowledge Graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2657–2666.

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. 2021a. Bringing Light into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models under a Unified Framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 12 (2021), 8825–8845.

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. 2021b. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research* 22, 82 (2021), 1–6.

Stephan Baier, Yunpu Ma, and Volker Tresp. 2017. Improving Visual Relationship Detection Using Semantic Modeling of Scene Descriptions. In *International Semantic Web Conference*. Springer, 53–68.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2018. Accelerating Neural Architecture Search Using Performance Prediction. In *The Sixth International Conference on Learning Representations (Workshop)*.

Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019. Tucker: Tensor Factorization for Knowledge Graph Completion. In *Proceedings of the 2019 Con-*

ference on Empirical Methods in Natural Language Processing. 5185–5194.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, Vol. 24.

Patrick Betz, Luis Galarraga, Simon Ott, Christian Meilicke, Fabian M Suchanek, and Heiner Stuckenschmidt. 2024. PyClause-simple and Efficient Rule Handling for Knowledge Graphs. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence: Demo Track*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. 1247–1250.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2014. A Semantic Matching Energy Function for Learning with Multi-Relational Data: Application to Word-Sense Disambiguation. *Machine Learning* 94 (2014), 233–259.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-Relational Data. In *Advances in Neural Information Processing Systems*, Vol. 26. 2787–2795.

Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. 2020. LibKGE - A Knowledge Graph Embedding Library for Reproducible Research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 165–174.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2021. Autoregressive Entity Retrieval. In *The Ninth International Conference on Learning Representations*.

Qiushi Cao, Cecilia Zanni-Merk, Ahmed Samet, Christoph Reich, François De Bertrand De Beuvron, Arnold Beckmann, and Cinzia Giannetti. 2022. KSPMI: A Knowledge-Based System for Predictive Maintenance in Industry 4.0. *Robotics and Computer-Integrated Manufacturing* 74 (2022), 102281.

Alberto Cattaneo, Daniel Justus, Harry Mellor, Douglas Orr, Jerome Maloberti, Zhenying Liu, Thorin Farnsworth, Andrew Fitzgibbon, Blazej Banaszewski, and Carlo Luschi. 2022. BESS: Balanced Entity Sampling and Sharing for Large-Scale Knowledge Graph Completion. *arXiv preprint arXiv:2211.12281* (2022). arXiv:2211.12281

Linlin Chao, Jianshan He, Taifeng Wang, and Wei Chu. 2021. PairRE: Knowledge Graph Embeddings via Paired Relation Vectors. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International*

Joint Conference on Natural Language Processing (Volume 1: Long Papers), Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 4360–4369.

Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. 2021. HittER: Hierarchical Transformers for Knowledge Graph Embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 10395–10407.

Alla Chepurova, Aydar Bulatov, Yurii Kuratov, and Mikhail Burtsev. 2023. Better Together: Enhancing Generative Knowledge Graph Completion with Language Models and Neighborhood Information. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 5306–5316.

Louis Clouatre, Philippe Trempe, Amal Zouaq, and Sarath Chandar. 2021. MLMLM: Link Prediction with Mean Likelihood Masked Language Model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 4321–4331.

Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. 2019. AmpliGraph: A Library for Representation Learning on Knowledge Graphs.

Gabor Csardi, Tamas Nepusz, et al. 2006. The Igraph Software Package for Complex Network Research. *InterJournal, complex systems* 1695, 5 (2006), 1–9.

Daniel Daza, Michael Cochez, and Paul Groth. 2021. Inductive Entity Representations from Text via Link Prediction. In *Proceedings of the Web Conference 2021*. 798–808.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing Factual Knowledge in Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 6491–6506.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d Knowledge Graph Embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024.

The Faiss Library. *arXiv preprint arXiv:2401.08281* (2024). arXiv:2401.08281

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of machine learning research* 12, 7 (2011).

Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofía Samaniego, Ying Xiao, et al. 2022. Twhin: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2842–2850.

Patrick Esser, Robin Rombach, and Bjorn Ommer. 2021. Taming Transformers for High-Resolution Image Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12873–12883.

Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.

Thibault Févry, Livio Baldini Soares, Nicholas Fitzgerald, Eunsol Choi, and Tom Kwiatkowski. 2020. Entities as Experts: Sparse Memory Access with Entity Supervision. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 4937–4951.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. In *Proceedings of the 22nd International Conference on World Wide Web*. 413–422.

Mikhail Galkin, Max Berrendorf, and Charles Tapley Hoyt. 2022. An Open Challenge for Inductive Link Prediction on Knowledge Graphs. *arXiv preprint arXiv:2203.01520* (2022). arXiv:2203.01520

Mikhail Galkin, Etienne Denis, Jiapeng Wu, and William L Hamilton. 2021. NodePiece: Compositional and Parameter-Efficient Representations of Large Knowledge Graphs. In *The Ninth International Conference on Learning Representations*.

Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 69–77.

Gene Ontology Consortium. 2004. The Gene Ontology (GO) Database and Informatics Resource. *Nucleic acids research* 32, suppl_1 (2004), D258–D261.

Ronald L Graham. 1966. Bounds for Certain Multiprocessing Anomalies. *Bell system technical journal* 45, 9 (1966), 1563–1581.

Alex Graves. 2012. Sequence Transduction with Recurrent Neural Networks. *arXiv preprint arXiv:1211.3711* (2012). arXiv:1211.3711

Robert Gray. 1984. Vector Quantization. *IEEE Assp Magazine* 1, 2 (1984), 4–29.

Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. 2017. Knowledge Transfer for Out-of-Knowledge-Base Entities: A Graph Neural Network Approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 1802–1808.

Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding Logical Queries on Knowledge Graphs. In *Advances in Neural Information Processing Systems*, Vol. 31. 2030–2041.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates Inc., Red Hook, NY, USA, 1025–1035.

Chi Han, Qizheng He, Charles Yu, Xinya Du, Hanghang Tong, and Heng Ji. 2023. Logical Entity Representation in Knowledge-Graphs for Differentiable Rule Learning. In *The Eleventh International Conference on Learning Representations*.

Jiawei Han. 2009. Mining Heterogeneous Information Networks by Exploring the Power of Links. In *12th International Conference on Discovery Science*. Springer, 13–30.

Kelvin Han and Claire Gardent. 2023. Multilingual Generation and Answering of Questions from Texts and Knowledge Graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 13740–13756.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. OpenKE: An Open Toolkit for Knowledge Embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 139–144.

John Hewitt, John Thickstun, Christopher Manning, and Percy Liang. 2023. Backpack Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 9103–9125.

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The Curious Case of Neural Text Degeneration. In *International Conference on Learning*

Representations.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*, Vol. 35.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge Graph Embedding Based Question Answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, New York, NY, USA, 105–113.

Andreea Iana, Mehwish Alam, and Heiko Paulheim. 2022. A Survey on Knowledge-Aware News Recommender Systems. *Semantic Web Journal* (2022).

Dora Jambor, Komal Teru, Joelle Pineau, and William L Hamilton. 2021. Exploring the Limits of Few-Shot Link Prediction in Knowledge Graphs. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2816–2822.

Kevin Jamieson and Ameet Talwalkar. 2016. Non-Stochastic Best Arm Identification and Hyperparameter Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. PMLR, 240–248.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product Quantization for Nearest Neighbor Search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. 2021. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

Fan Jiang, Tom Drummond, and Trevor Cohn. 2023. Don't Mess with Mister-in-between: Improved Negative Search for Knowledge Graph Completion. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. 1810–1824.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

Minki Kang, Jinheon Baek, and Sung Ju Hwang. 2022. KALA: Knowledge-augmented Language Model Adaptation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 5144–5167.

George Karypis and Vipin Kumar. 1998. Multilevel k-Way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed computing* 48, 1 (1998), 96–129.

Seyed Mehran Kazemi and David Poole. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *Advances in Neural Information Processing Systems*, Vol. 32. 4289–4300.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *The Second International Conference on Learning Representations*.

Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *The Third International Conference on Learning Representations*.

Thomas P Kirkman. 1847. On a Problem in Combinations. *Cambridge and Dublin Mathematical Journal* 2 (1847), 191–204.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. In *The Eighth International Conference on Learning Representations*.

Adrian Kochsiek and Rainer Gemulla. 2021. Parallel Training of Knowledge Graph Embedding Models: A Comparison of Techniques. *Proceedings of the VLDB Endowment* 15, 3 (2021), 633–645.

Adrian Kochsiek and Rainer Gemulla. 2023. A Benchmark for Semi-Inductive Link Prediction in Knowledge Graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 10634–10643.

Adrian Kochsiek, Fritz Niesel, and Rainer Gemulla. 2022. Start Small, Think Big: On Hyperparameter Optimization for Large-Scale Knowledge Graph Embeddings. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 138–154.

Adrian Kochsiek, Apoorv Saxena, Inderjeet Nair, and Rainer Gemulla. 2023. Friendly Neighbors: Contextualized Sequence-to-Sequence Link Prediction. In *Proceedings of the 8th Workshop on Representation Learning for NLP*.

Bhushan Kotnis and Vivi Nastase. 2017. Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs. *arXiv preprint arXiv:1708.06816* (2017). arXiv:1708.06816

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. 2018. Canonical Tensor Decomposition for Knowledge Base Completion. In *Proceedings of 35th International Conference on Machine Learning*. PMLR, 2863–2872.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-Biggraph: A Large Scale Graph Embedding System. *Proceedings of Machine Learning and Systems* 1 (2019), 120–131.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al.

2020. Retrieval-Augmented Generation for Knowledge-Intensive Nlp Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. *Proceedings of Machine Learning and Systems* 2 (2020), 230–246.

Shiyang Li, Yifan Gao, Haoming Jiang, Qingyu Yin, Zheng Li, Xifeng Yan, Chao Zhang, and Bing Yin. 2023. Graph Reasoning for Question Answering with Triplet Retrieval. In *Findings of the Association for Computational Linguistics: ACL 2023*. 3366–3375.

Shaobo Li, Xiaoguang Li, Lifeng Shang, Cheng-Jie Sun, Bingquan Liu, Zhenzhou Ji, Xin Jiang, and Qun Liu. 2022b. Pre-Training Language Models with Deterministic Factual Knowledge. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 11118–11131.

Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2021. Mfes-Hb: Efficient Hyperband with Multi-Fidelity Quality Measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8491–8500.

Zonglin Li, Ruiqi Guo, and Sanjiv Kumar. 2022a. Decoupled Context Processing for Context Augmented Language Modeling. In *Advances in Neural Information Processing Systems*, Vol. 35. 21698–21710.

Robert Logan, Nelson F. Liu, Matthew E. Peters, Matt Gardner, and Sameer Singh. 2019. Barack's Wife Hillary: Using Knowledge Graphs for Fact-Aware Language Modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 5962–5971.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2015. Yago3: A Knowledge Base from Multilingual Wikipedias. In *Conference on Innovative Data Systems Research*.

Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckeschmidt. 2024. Anytime Bottom-up Rule Learning for Large-Scale Knowledge Graph Completion. *The VLDB Journal* 33, 1 (2024), 131–161.

Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-up Rule Learning for Knowledge Graph Com-

pletion.. In *IJCAI*. 3137–3143.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. 2023. Finite Scalar Quantization: VQ-VAE Made Simple. In *The Twelfth International Conference on Learning Representations*.

Sameh K Mohamed, Aayah Nounu, and Vít Nováček. 2019. Drug Target Discovery Using Knowledge Graph Embeddings. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 11–18.

Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. 2021. Marius: Learning Massive Graph Embeddings on a Single Machine. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 533–549.

Sebastian Monka, Lavdim Halilaj, and Achim Rettinger. 2022. A Survey on Visual Transfer Learning Using Knowledge Graphs. *Semantic Web Journal* 13, 3 (2022), 477–510.

Sebastian Monka, Lavdim Halilaj, Stefan Schmid, and Achim Rettinger. 2021. Learning Visual Models Using a Knowledge Graph as a Trainer. In *International Semantic Web Conference*. Springer, 357–373.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to Compress Prompts with Gist Tokens. In *Advances in Neural Information Processing Systems*, Vol. 37.

Kenton Murray and David Chiang. 2018. Correcting Length Bias in Neural Machine Translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*. Association for Computational Linguistics, Brussels, Belgium, 212–223.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2015. A Review of Relational Machine Learning for Knowledge Graphs. *Proc. IEEE* 104, 1 (2015), 11–33.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data. In *Proceedings of the 28th International Conference on Machine Learning*. PMLR, 809–816.

Fabio Petroni, Tim Rocktäschel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Michihiro Yasunaga, Haitian Sun, Dale Schuurmans, Jure Leskovec, and Denny Zhou. 2021. LEGO: Latent Execution-Guided Reasoning for Multi-Hop Question Answering on Knowledge Graphs. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 8959–8970.

Alexander Renz-Wieland, Rainer Gemulla, Steffen Zeuch, and V. Markl. 2020. Dynamic Parameter Allocation in Parameter Servers. *Proceedings of the VLDB Endowment* 13, 11 (2020), 1877–1890.

Alexander Renz-Wieland, Andreas Kieslinger, Robert Gericke, Rainer Gemulla, Zoi Kaoudi, and Volker Markl. 2023. Good Intentions: Adaptive Parameter Management via Intent Signaling. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2156–2166.

Peter N Robinson, Sebastian Köhler, Sebastian Bauer, Dominik Seelow, Denise Horn, and Stefan Mundlos. 2008. The Human Phenotype Ontology: A Tool for Annotating and Analyzing Human Hereditary Disease. *The American Journal of Human Genetics* 83, 5 (2008), 610–615.

Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2021. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *ACM Transactions on Knowledge Discovery from Data* 15, 2 (2021), 1–49.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2020. You {CAN} Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. In *The Eighth International Conference on Learning Representations*.

Daniel Ruffinelli and Rainer Gemulla. 2024. Beyond Link Prediction: On Pre-Training Knowledge Graph Embeddings. In *Proceedings of the 9th Workshop on Representation Learning for NLP*.

V Sanh. 2019. DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter.. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing @ NeurIPS*, Vol. 5.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-Sequence Knowledge Graph Completion and Question Answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2814–2828.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving Multi-Hop Question Answering over Knowledge Graphs Using Knowledge Base Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 4498–4507.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *Extended Semantic Web Conference*. Springer, 593–607.

Stephen B Seidman. 1983. Network Structure and Minimum Degree. *Social networks* 5, 3 (1983), 269–287.

Haseeb Shah, Johannes Villmow, Adrian Ulges, Ulrich Schwanecke, and Faisal Shafait. 2019. An Open-World Extension to Knowledge Graph Completion Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3044–3051.

Baoxu Shi and Tim Weninger. 2018. Open-World Knowledge Graph Completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32. 1957–1964.

Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A Survey of Heterogeneous Information Network Analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

Kensen Shi, David Bieber, and Charles Sutton. 2020. Incremental Sampling without Replacement for Sequence Models. In *International Conference on Machine Learning*. PMLR, 8785–8795.

Harkanwar Singh, Soumen Chakrabarti, PRACHI JAIN, Sharod Roy Choudhury, et al. 2021. Multilingual Knowledge Graph Completion with Joint Relation and Entity Alignment. In *3rd Conference on Automated Knowledge Base Construction*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and Permuted Pre-Training for Language Understanding. In *Advances in Neural Information Processing Systems*, Vol. 33. 16857–16867.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*. 697–706.

Haitian Sun, Andrew O. Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W. Cohen. 2021. Faithful Embeddings for Knowledge Base Queries. In *Advances in Neural Information Processing Systems*, Vol. 33.

Kai Sun, Yifan Ethan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. 2023. Head-to-Tail: How Knowledgeable Are Large Language Models (LLM)? AKA Will LLMs Replace Knowledge Graphs? *arXiv preprint arXiv:2308.10168* (2023). arXiv:2308.10168

Yizhou Sun and Jiawei Han. 2013. Mining Heterogeneous Information Networks: A Structural Analysis Approach. *ACM SIGKDD Explorations* 14, 2 (2013), 20–28.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *The Seventh*

International Conference on Learning Representations.

Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. 2020. A Re-evaluation of Knowledge Graph Completion Methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 5516–5522.

Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive Relation Prediction by Subgraph Reasoning. In *International Conference on Machine Learning*. PMLR, 9448–9457.

Kristina Toutanova and Danqi Chen. 2015. Observed versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and Their Compositionality*. 57–66.

Hung Nghiep Tran and Atsuhiro Takasu. 2022. MEIM: Multi-partition Embedding Interaction beyond Block Term Format for Efficient and Expressive Link Prediction. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence*. 2262–2269.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33rd International Conference on Machine Learning*. PMLR, 2071–2080.

Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. 2019. Autone: Hyperparameter Optimization for Massive Network Embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 216–225.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural Discrete Representation Learning. In *Advances in Neural Information Processing Systems*, Vol. 30. 6306–6315.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 31.

Blerta Veseli, Simon Razniewski, Jan-Christoph Kalo, and Gerhard Weikum. 2023. Evaluating the Knowledge Base Completion Potential of GPT. *Findings of the Association for Computational Linguistics: EMNLP 2023* (2023), 6432–6443.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. 2022. Phenaki: Variable Length Video Generation from Open Domain Textual Descriptions. In *The Tenth International Conference on Learning Representations*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2023. MariusGNN: Resource-efficient out-of-Core Training of Graph Neural Networks. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 144–161.

Bo Wang, Tao Shen, Guodong Long, Tianyi Zhou, Ying Wang, and Yi Chang. 2021c. Structure-Augmented Text Representation Learning for Efficient Knowledge Graph Completion. In *Proceedings of the Web Conference 2021*. 1737–1748.

Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. 2022. SimKGC: Simple Contrastive Knowledge Graph Completion with Pre-Trained Language Models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 4281–4294.

Peifeng Wang, Jialong Han, Chenliang Li, and Rong Pan. 2019. Logic Attention Based Neighborhood Aggregation for Inductive Knowledge Graph Embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7152–7159.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

Ruochen Wang, Xiangning Chen, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. 2021a. RANK-NOSH: Efficient Predictor-Based Architecture Search via Non-Uniform Successive Halving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10377–10386.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021b. KEPLER: A Unified Model for Knowledge Embedding and Pre-Trained Language Representation. *Transactions of the Association for Computational Linguistics* 9 (2021), 176–194.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural Text Generation with Unlikelihood Training. In *The Eighth International Conference on Learning Representations*.

Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation* 1, 2 (1989), 270–280.

Yuxiang Wu, Yu Zhao, Baotian Hu, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2022. An Efficient Memory-Augmented Transformer for Knowledge-Intensive NLP Tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 5184–5196.

Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation Learning of Knowledge Graphs with Entity Descriptions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30. 2659–2665.

Wenhan Xiong, Jingfei Du, William Yang Wang, and Veselin Stoyanov. 2020. Pretrained Encyclopedia: Weakly Supervised Knowledge-Pretrained Language Model. In *The Eighth International Conference on Learning Representations*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A Massively Multilingual Pre-Trained Text-to-Text Transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 483–498.

Muhammad Yahya, John G Breslin, and Muhammad Intizar Ali. 2021. Semantic Web and Knowledge Graphs for Industry 4.0. *Applied Sciences* 11, 11 (2021), 5110.

Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, et al. 2023. A Comprehensive Study on Text-Attributed Graphs: Benchmarking and Rethinking. In *Advances in Neural Information Processing Systems*, Vol. 36. 17238–17264.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *The Second International Conference on Learning Representations*.

Yilin Yang, Liang Huang, and Mingbo Ma. 2018. Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. KG-BERT: BERT for Knowledge Graph Completion. *arXiv preprint arXiv:1909.03193* (2019). arXiv:1909.03193

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big Bird: Transformers for Longer Sequences. *Advances in Neural Information Processing Systems* 33 (2020), 17283–17297.

Denghui Zhang, Manling Li, Yantao Jia, Yuanzhuo Wang, and Xueqi Cheng. 2017. Efficient Parallel Translating Embedding for Knowledge Graphs. In *Proceedings of the International Conference on Web Intelligence*. 460–468.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. 2019. Quaternion Knowledge Graph Embeddings. In *NeurIPS*.

Yongqi Zhang, Zhanke Zhou, Quanming Yao, and Yong Li. 2022. Efficient Hyperparameter Search for Knowledge Graph Embedding. In *Proceedings of the 60th*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2715–2735.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3848–3860.

Chenguang Zheng, Guanxian Jiang, Xiao Yan, Peiqi Yin, Qihui Zhou, and James Cheng. 2024. GE2: A General and Efficient Knowledge Graph Embedding Learning System. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. Dgl-Ke: Training Knowledge Graph Embeddings at Scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 739–748.

Zhaocheng Zhu, Shizhen Xu, Meng Qu, and Jian Tang. 2019. GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding. In *The World Wide Web Conference*. ACM, 2494–2504.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural Bellman-Ford Networks: A General Graph Neural Network Framework for Link Prediction. In *Advances in Neural Information Processing Systems*, Vol. 34. 29476–29490.

Daniel Zwillinger and Stephen Kokoska. 1999. *CRC Standard Probability and Statistics Tables and Formulae*. Crc Press.

List of Figures

List of Tables

List of Algorithms