Enhancing Link Prediction in Knowledge Graphs Through Pre-Informed Training

Sławomir Andrzej Męczyński[1], Daniel Daza[2], and Michael Cochez[3]

[1] Vrije Universiteit Amsterdam, The Netherlands
s.a.meczynski@student.vu.nl
[2] Vrije Universiteit Amsterdam, The Netherlands
d.dazacruz@vu.nl
[3] Vrije Universiteit Amsterdam, The Netherlands
m.cochez@vu.nl

Abstract. Knowledge Graphs (KG) represent a network of real-world entities and relationships between them. They enable a wide variety of applications, such as information retrieval and question answering. However, KGs evolve in time and are often incomplete. To address this, a variety of KG embeddings is employed. Given an incomplete KG, these machine learning models are used to predict missing edges. The prediction happens by asking the model to which node T it would connect a new edge with a label R if it starts at a node H. In this work we investigate whether such a model could benefit from knowing the labels of the edges that will likely be asked for. This involves two approaches - extending the training set and manipulating the loss function. Both strategies leverage a recommender system that employs a trie-based method called SchemaTree. Experimental results on FB15k-237 show faster early-stage convergence in terms of training steps, though final performance aligns with the baseline.

Keywords: Knowledge Graphs · Link Prediction · Pre-Informed Training · Graph Embedding.

1 Introduction

Knowledge Graphs (KGs) are structured, machine-readable representations of real-world information [31,6]. They are gaining prominence as frameworks of knowledge bases that, using graph-structured models, represent data and are used to reason and infer relevant information. Their ability to capture rich semantic relationships makes them a powerful foundation for a variety of AI and data science tasks.

KGs consist of nodes and edges. Nodes represent **entities**, such as places, organisations, and people. Edges, with given labels, are binary **relations** between nodes. They can be directed (in *Directed Edge-labeled Graphs* [21] - in short *del graphs*, also known as *multi-relational graphs* [9]) or undirected. Consider the following example: Head Node *M S. Curie* -> Directed Edge with label *born_ in* -> Tail Node *Warsaw*. This triple could be a part of KG, as illustrated in Fig. 1.

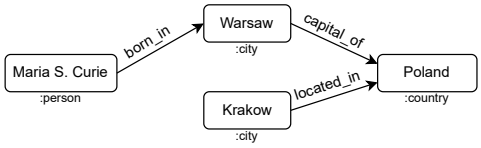

Fig. 1. Example illustrating a knowledge base fragment with nodes (entities), edges (relations).

If such a defined KG fragment was part of an ontology (domain-specific specification with predefined restrictions), the entities (nodes) would also have classes (categories) *person*, *city*, and *country* (in the Fig. 1 shown below the entities). A standard data structure based on *del graphs* [21] is RDF - *Resource Description Framework* [44]. In this data model, there are three types of nodes: *IRIs* (Internationalised Resource Identifiers that redirect to the World Wide Web), *literals* (strings, integers, dates, etc.), and *blank nodes* (used to denote the existence of an entity). Besides *del graphs*, there are other similar types, such as *heterogeneous graphs* [22] and property graphs [5].

1.1 The Evolution and Importance of Knowledge Graphs

The idea of KGs has been used in the literature since at least 1972 [34]. In the 1980s, the researchers at the University of Groningen and the University of Twente introduced the term KG to formally describe their systems that integrated knowledge from natural language [17,29]. In 2012, Google introduced the KG as a semantic enhancement of Google's search engine [36], which revolutionised related research fields. Since that time, the application of KG has been evolving and used in various areas, including healthcare, education, ICT, science and engineering, finance, society and politics, and travel [3]. Some of the notable examples of KGs are DBPedia [7], Yago[37], and FreeBase [11]. In our research, we use FB15k237 [39], which is a subset of Freebase (explained in subsection 2.3). There are also various industry-based KGs used by Airbnb, Amazon, eBay, Facebook, IBM, LinkedIn, Microsoft, Uber, and many more [21]. This database approach - KG - is widely adopted across numerous organizations and researchers because of its concise and intuitive representation of many domains, where edges and paths (sequence of edges) express various, often complex, relationships between real-world entities [6].

KGs that have the *Closed World Assumption* (CWA) assume that what is not known (if there are no such nodes and edges) is assumed to be false. However, since real-world KGs often have the *Open World Assumption* (OWA), where it is possible for the relation to exist without being explicitly described by the graph [8], they often suffer from **incompleteness**. Even the richest KGs do not contain all the information needed; for example, in FreeBase, over 70% of person entities have no known place of birth, and over 99% have no known ethnicity [16,43]. Hence, to identify new facts, we need the process called *Knowledge*

Graph Completion or *Knowledge Graph Augmentation*. There are two common approaches: extracting new facts from external sources and inferring them based on patterns in a given KG. The latter, called *Link Prediction*, is the area of our research.

1.2 Link Prediction

Link Prediction (LP) is a research area that makes use of a wide range of techniques, including similarity-based indices, probabilistic methods, and dimensionality reduction approaches [25]. This essential for completing and extending existing graphs task is widely studied in KG research [48,24,42]. In recent years, LP has benefited from different Machine Learning (ML) and Deep Learning (DL) techniques. Most of such models use a low-dimensional representation of KGs called *Knowledge Graph Embeddings*. Such techniques enable ML models to reason over and predict new facts - finding missing links (in static networks) or predicting the likelihood of future links (in dynamic networks) [25]. In our research, we focus on static networks with a possible extension for dynamic networks in the future.

To reason about LP, consider the KG as a tuple. Assume that the KG is directed and labeled with the Open World Assumption: $\mathcal{KG} = (\mathcal{E}, \mathcal{R}, \mathcal{G})$, where:

- $\mathcal{E}$ is a set of nodes (entities),
- $\mathcal{R}$ is a set of labels (representing relations), also called properties, predicates, or, in the context of Wikidata, P-values
- $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is a set of directed edges (relations) representing facts connecting pairs of entities.

In such a configuration, let each fact be a triple $\langle h, r, t \rangle$, where $h \in \mathcal{E}$ is the head entity, $r \in \mathcal{R}$ is the relation with a label, and $t \in \mathcal{E}$ is the tail entity.

In LP, we explore the existing facts (nodes and edges with labels) to infer the missing facts. In other words, most often we are guessing the missing tail node $\langle h, r, ? \rangle$, head node $\langle ?, r, t \rangle$, and sometimes the relation $\langle h, ?, t \rangle$. However, we focus on the first two, as relation prediction is more challenging and less informative in practice because the set of relations is usually much smaller than the sets of entities. We will call the known entity a *source* and the entity that is predicted a *target*.

1.3 Graph Embeddings

Graph Embeddings are numerical vectors that are used to represent entities and relations from a KG in a continuous feature space. Those low-dimensional vector spaces not only preserve information about existing triples in the KG, but also about new triples that could be predicted [23]. They enable ML models to reason about the graph's structure. *"They are learned automatically, based on how the corresponding elements occur and interact with each other in datasets representative of the real world."* [31]. Graph Embeddings capture the semantic

structure of the original graph by encoding how entities and relations interact with each other. This allows ML models to reason about the graph and predict missing links based on these learned patterns. KG Embeddings generally have two types - node embedding and relation embedding. In all of them, the data is divided into the training set G_{train}, validation set G_{valid} and the test set G_{test}, often using an 80/10/10 or a similar split.

Embedding models There is a variety of embedding-based approaches in LP, from symbolic learning, through supervised to unsupervised learning [21]. In *symbolic learning*, the model uses logical formulae - rules and axioms. In *supervised learning*, the graph structure is directly leveraged through Graph Neural Networks (GNN). In *unsupervised learning*, common strategies include *graph analytics*, which are typically used to identify communities or clusters and to detect central nodes and edges. There is also a *self-supervision* that learns a low-dimensional numerical model of elements of a KG, called *knowledge graph embeddings* (KGE). This embedding model is a baseline of our research; hence, in this subsection, we explain the idea behind such models and describe ComplEx [40] - a popular implementation, which is also used in our research.

The primary goal of KGEs is to create a dense representation of the graph in a continuous, low-dimensional vector space that can later be used in ML tasks. The number of dimensions d is fixed and usually takes low values, between 50 and 1000. In most cases, the KGE has an *entity embedding* for each node and a *relation embedding* for each edge label. They form vectors that preserve latent structures in the KG. There are several techniques to form and train KGEs, such as *translational models* [12] (where relations are interpreted as transformations that map subject entities to object entities), *tensor decomposition models* [45] (which uncover hidden patterns that approximate the underlying structure of the graph), *neural models* [15] (using neural neworks), and *language models* [30] (using word embedding).

In our research, we focus on the tensor factorization model - **ComplEx**. Trouillon, T et al. (2016) [40], through latent factorization, managed to build the ComplEx model - an alternative approach outperforming standard link prediction benchmarks. This comparatively simple embedding model [40] makes use of the Hermitian dot product - the complex counterpart of the standard dot product between real vectors. Such a method was potentially adaptable to our approach of pre-informed training methods described in the next sections; hence, we performed our model mostly using ComplEx.

1.4 Literature review - pre-training

Traditional embedding-based methods, such as TransE and ComplEx, have improved tail entity prediction but struggle to generalize to unseen entities during testing [20]. Embedding-based methods, such as DistMult [26], successfully represent the KG as vectors; however, they fail with unseen entities or relations during testing [49]. Such models use fixed representations, but ignore context-specific variations [46]. In general, the notable limitations of embedding-based

models, such as TransE and ComplEx, struggle with the generalization of unseen entities with their relations due to their dependence on predefined embeddings. This limits the adaptability of new KG structures.

To counteract such problems, a solution could be to provide the embedding model with prior knowledge to guide training. The process, which we will call *pre-training*, is a strategy that has been partially used in many knowledge representation ML problems, including negative sampling. This is done to *teach* the model to discriminate between true (positives) and false (negatives) statements [10]. An example, used by von Rueden L. et al. (2022) [32], integrated the pre-informed step to enhance training by injecting prior knowledge. It was done by transforming the graph representation into a small, condensed data set of knowledge prototypes. This process sped up the training, improved generalization, and increased performance. Although this was done using knowledge prototypes and tested mainly in image classification and regression, the idea behind incorporating prior knowledge during pre-training to guide the model toward more relevant features aligns closely with our approach. In our case, the *pre-informed* knowledge is in the form of edge labels that are likely to be queried.

Another approach that uses pre-informed knowledge, but especially for link prediction, was introduced by Jain, N. et al. (2021) [23] and made use of generating *more suitable* negative sampling. Their *ReasonKGE* improved the accuracy of embeddings using ontological reasoning. It was done by incorporating negative samples retrieved from the process of dynamically identifying inconsistent predictions via symbolic reasoning. Additional triples were inferred using description logic rules (from subclass, subproperty, domain, and range axioms). It resulted in an improvement of accuracy compared to the state-of-the-art.

On the other hand, Bernardi, A. et al. (2024) [10] constructed a strategy for generating corruptions during training, while preserving the domain and range of relations. This ontology-based sampling involves randomly replacing either the subject or object in a positive triple with another entity. This resulted in substantial improvement (+10% *MRR*) for standard benchmark datasets and over +150% *MRR* for a larger ontology-backed dataset. However, in our work, instead of generating negatives, we generate potential positives.

1.5 Research Objectives - Motivation

As we showed in subsection 1.4, in recent years, most LP methods in KG have largely relied on similar core methods that improve the models but do not entirely capture the whole dependencies in KG that are needed for LP. Hence, in this research, we introduce a novel **pre-informed** strategy that incorporates knowledge about the labels of edges that are likely to be queried. Consider the following example: given an entity like *Amsterdam*, queries involving relation *capital_of, birthplace_of* are more likely than *acted_in_movie*, or *married_with*. Such a scenario is presented in the figure Fig. 2.

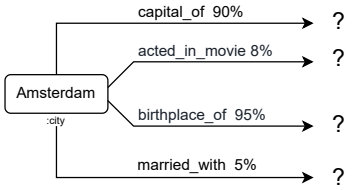

Fig. 2. Example illustrating an entity *Amsterdam* with possible edge labels and their probabilities that can be used in the training phase to prioritize likely queried edges.

The information (probabilities of a label occurrence for a given entity) is sourced from a recommender system that predicts the labels of entities (edge relations) that are often observed together. This can be a potentially valuable source of auxiliary knowledge by prioritizing patterns that are more statistically *relevant*.

Accordingly, our research question is: **Can an LP model benefit from knowing in advance which edge labels are likely to be queried?** We want to test the hypothesis that such a pre-informed training strategy, where training prioritizes more relevant edge labels, results in statistically significant improvements for LP performance measured by *Hits@K* and *MRR* metrics compared to baseline embedding models, including ComplEx.

2 Methodology

In this section, we describe the evaluation metrics we use, the Recommender System (SchemaTree - Maximum-Likelihood Property Recommendation), and a high-level overview of the implementation of two approaches (which are described in detail in the next section). Our implementation is based on PyKEEN[4] - a Python library that is specifically designed for training and evaluating knowledge graph embedding models. The training process is done on the supercomputer provided by the Dutch SURF service - Snellius [38]. The code, together with the Recommender System, is available at GitHub [27]. All experiments are conducted with ethical considerations, ensuring transparency, reproducibility, and responsible use of computational resources.

Due to preserving consistency, we refer to an edge (any connection) with a label as a *relation* (in the context of KG itself) and as a *property* (in the context of the Recommender System [13], described in subsection 2.1). However, both refer to the same concept of a link between two nodes.

2.1 Recommender System

For the pre-informed training, we need to obtain properties that are likely to be queried in the test set. Specifically, for each training entity, we aim to obtain a

[4] `https://github.com/pykeen/pykeen`

ranked set of candidate properties with associated probabilities. This estimation is performed by the Recommender System.

The Recommender System is the core base from which we take the information of the labels of likely queried edges. There are several data-driven property recommendation systems that have been introduced [1,2,35]. The one we use is SchemaTree - Maximum-Likelihood Property Recommendation for Wikidata, introduced by Lars C. Gleim et al. (2020) [19]. We chose this system because it outperforms the state-of-the-art Wikidata PropertySuggester that uses an association rule-based approach [47]. Also, this system's average rank of the first relevant recommendation was reduced by 71% compared to the mentioned Wikidata PropertySuggester. We use an updated version of this system [13].

The approach of the Recommender System uses a novel **trie-based** method that can efficiently learn and represent property set probabilities in RDF graphs. It can also be applied to other KGs due to its generalizability. Additionally, it has an extension of adding *"type information to improve recommendation precision and introduce backoff strategies which further increase the performance of the initial approach for entities with rare property combinations"* [19]. However, we implemented our own *type information system* (described in the subsection 3.2) due to the original system technical issues.

The core idea of the system is to use the **maximum likelihood** estimation to determine the most likely property a given the already observed properties of an entity. In other words, we find the property a that is most often observed together with the properties that the given entity already has. Then, we extend the definition to return a list of k properties with the maximum likelihood probabilities. At the end, this list is sorted by the probability. This process, where the property that we are looking for is $\hat{a} \in \mathcal{A} \setminus S$, is shown in Equation (1)

$$\hat{a} = \arg \max_{a \in (\mathcal{A} \setminus S)} \frac{P(\{a, s_1, \ldots, s_n\})}{P(\{s_1, \ldots, s_n\})} \tag{1}$$

Where:

- $\mathcal{A}$: The set of all properties in the KG.
- $S = \{s_1, \ldots, s_n\}$: The set of properties currently assigned to the entity E.
- $a \in (\mathcal{A} \setminus S)$: A candidate property not yet assigned to E.
- $P(\{a, s_1, \ldots, s_n\})$: The probability that a randomly selected entity has at least the properties $a, s_1, \ldots, s_n$.
- $P(\{s_1, \ldots, s_n\})$: The probability that a randomly selected entity has at least the properties $s_1, \ldots, s_n$.

The Recommender System is mostly based using the GO language and is queried using JSON format

2.2 Loss Function and Evaluation Metrics

The embeddings are initialized randomly and then adjusted in the training process. As we want to maximize the plausibility of the true facts and minimize it for

the false facts, the loss function needs to gather the scores for all facts in G_{train}. The combination of both positive and negative samples is often achieved by applying a triplet-based loss function. Popular loss functions for LP include Multi-class Negative Log-Likelihood, Pairwise Ranking Loss (margin-based), and Self-adversarial Negative Sampling Loss [31]. In our research, we use Cross-Entropy Loss, as it is well-suited for binary classification with sigmoid scoring - which aligns with ComplEx's link prediction objective.

The evaluation is done by performing head h and tail t prediction of the triples $\langle h, r, t \rangle$ in G_{test}. For each prediction, the rank of the correct entity is computed relative to all possible candidates. *"Ranks can be computed in two largely different settings, called raw and filtered scenarios, depending on how other valid answers outranking the target one are handled."* [31]. If a predicted entity produces a fact contained in G, it can still be considered a valid answer, even if it differs from the originally held target entity. Here are the two options:

- Raw scenario - valid nodes outscoring the target one are considered as mistakes and contribute to the rank computation.
- Filtered Scenario - valid nodes outscoring the target one are not considered as mistakes, and they are skipped in the rank computation

In our research, we use the filtered scenario. We use the metric *Hits@K*, which measures how often the correct answer appears in the top k predictions of our model. For instance, *Hits@3 = 0.60* means that the correct node was ranked in the first three predictions in 60% of the test cases. We aim to have the highest percentage across all values of k, ideally also for low k values. Mathematically, *Hits@K* is presented in Equation (2).

$$\textit{Hits@K} = \frac{|\{q \in Q \ : \ \text{rank}(q) \leq K\}|}{|Q|} \tag{2}$$

Where:

- Q: the set of all queries (triples in the test set)
- $\text{rank}(q)$: the rank of the correct entity for query q
- K: the top-K threshold
- The numerator counts correct answers ranked in the top K.

The score will always be between 0 and 1, where higher values mean better results. In our research, we use 1, 3, and 10 for the K threshold.

The rank that we use is the position of the correct entity in the sorted list of predictions obtained in the test phase. Hence, we aim to have the lowest rank. However, sometimes there are multiple entities that have the same score as the target one. This phenomenon is called a *tie* and there are several strategies (*tie-breaking policies*) for how to handle it, such as *max, average, original, random, and max* [31]. These strategies can yield varying results, so to ensure consistency we clearly state our chosen approach - *average* (PyKEEN's realistic).

Once the individual ranks in the test phase are calculated, in LP, there are common metrics for computing the global scores. In our research we use:

– Mean Rank (MR):

$$MR = \frac{1}{|Q|} \sum_{q \in Q} \text{rank}(q) \tag{3}$$

Where Q is the set of all test queries, and $\text{rank}(q)$ denote the rank assigned to the correct entity for a given query $q \in Q$. Hence, this metric calculates the average of obtained ranks. However, this is very sensitive to outliers, as explained by Nickel et al. (2016) [28]

– Mean Reciprocal Rank (MRR):

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}(q)} \tag{4}$$

Here, Q is also the set of all test queries, and $\text{rank}(q)$ denotes the rank assigned to the correct entity for a given query $q \in Q$. Hence, this metric returns the average of the inverse of the obtained ranks, as explained by Ellen M. Voorhees (1999) [41]

In both metrics, the perfect score is 1; however, MR ranges from 1 to $|E|$ (the number of entities) and MRR is always from 0 to 1. In our research, we use both MR and MRR to evaluate the models. PyKEEN allows getting optimistic, realistic, and pessimistic results for head, tail, and both separately. In the paper, we only present realistic results for both, although the differences are minimal.

2.3 Data

The data we use is FB15k-237[5]. This dataset was introduced by Toutanova and Chen (2015) [39] and is an adaptation from the Freebase KB, called FB15K [12]. The original FB15K consists of a large number of triples with inverses that caused leakage from the training to testing and validation splits. The new FB15k-237 has removed near-duplicate or inverse-duplicate relations, ensuring that the testing and evaluation datasets do not have an inverse relation test leakage. We also chose the dataset due to its comprehensive structure and wide usage in benchmarking LP models. This new dataset's latent feature models *"substantially outperform the observed feature models"* [39]. FB15k-237 contains 310,116 triples with 14,541 entities and 237 relation types. It is built in the PyKEEN library, see table 3 in the appendix.

To prepare for the later implementation, we analyzed the structure of the KG: 1,126 entities appear only as heads and 724 only as tails. Most entities have few or a moderate number of properties (see Appendix, Table 4).

3 Experiments

Given the Recommender System's recommendations, here we describe how they are used for the training and to conduct the experiments. Firstly, we choose

[5] `https://paperswithcode.com/dataset/fb15k-237`

a benchmark dataset described in subsection 2.3; then we conduct the Recommender System analysis, and finally, we test two pre-informed training approaches. The methods are functionally different - one incorporates a set of synthetic triples, and the second manipulates the loss function. Additionally, in the appendix, we include the hyperparameters' influence.

In our research, we use hyperparameters that were found to be the most optimal by Ruffinelli, D. et al. (2020) [33]. They also used the ComplEx model and FB15k-237 dataset, and their results are presented in the Appendix, Table 2.

3.1 Recommendations analysis

Before the actual experiments, we perform an analysis of the recommendations in order to find the optimal threshold used to keep recommended properties depending on their probability. First, we compute the set of recommended relations (source entities with their recommended properties) provided by the Recommender System. We perform that multiple times with different probability thresholds. Next, we compute the set of relations (source entities with their recommended properties) in the validation set. Precision, for each entity, is then the number of its recommended properties that the entity also has in the validation set (true positives) with respect to the total number of all its recommended properties. Recall is the same, but with respect to the total number of its properties in the validation set. Then, we perform macro-averaging - taking the mean across all entities. The results are shown in Fig. 3.

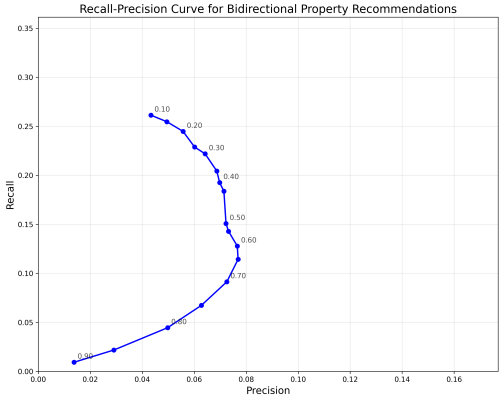

Fig. 3. Proportion of generated triples that are actually present in the validation set. Each dot represents the probability threshold.

The shape in Fig. 3 shows that once the recall decreases, the precision increases. However, at a certain threshold (around 0.4), the precision starts to decrease too, as a consequence of the Recommender System architecture. The relation types of an entity X included in the Recommender System query are excluded from possible recommended relation types of the entity X. Hence, even if there is a high possibility of reoccurrence of the same relation type, the model dismisses that. The model simply cannot recommend true positives that already exist. When the threshold is very high, the most confident predictions are the

ones that are *forbidden*, and the model is forced to recommend second-best predictions (which might be out of the threshold scope).

Based on this, we calculate the probability threshold (0.25) at which the F1 score is highest (0.095), and use it in the actual experiments. This yelds high accuracy of the extended models; however, the highest accuracy is achieved by manipulating other factors and functionalities (described in the next subsection).

We also evaluate how many recommended properties appear in the test set. The results are that 93,9% are within the test set. This implies that about 6% of recommended properties appear only in the training set (and possibly in the validation set), which may reduce the recommender's effectiveness.

3.2 Bidirectional artificial triples

This first approach aims to extend the embedding model by an artificial set of triples. The new triples are inferred by the Recommender System and progressively added to the training set. Once the training set is extended, we run the ComplEx model embedding on hyperparameters taken from Ruffinelli, D. et al. (2020) [33] and compare with baseline ComplEx trained on the original training set.

The process begins by building a schema tree from a TSV file, where each line contains the full set of properties for a single entity. The properties are given prefixes *I:* for incoming and *O:* for outgoing relations. The schema is then used to calculate the probability of new likely labels. It is done by querying the Recommender System with sets of all training properties of each training entity (the same way the schema was trained). Each set represents a given entity, so the recommender returns a set of possible labels for this entity. We sort them by likelihood, with the most probable on top. Then, we choose only the first $x\%$ recommendations (we call it *probability threshold*). At the end, for each chosen recommendation, we add a new triple to the training set.

The new triples are added in the following way: for each chosen recommendation of an entity X, add a relation Y with an artificial entity Y'. The relation can be either outgoing, when the original X is a head entity and Y' is a tail entity, or incoming, when Y' is a head entity and X is a tail entity. To prevent semantic drift (we introduce entities that are not used in validation or test phases), the new entities, like Y', are assigned to a particular relation type. This is done to ensure that the embedding model learns meaningful embeddings for the new entities by assigning them consistently to specific relation types rather than introducing entirely new entities each time. A visualization is presented in the Fig. 4.

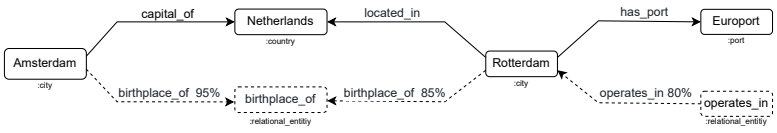

Fig. 4. Example illustrating the creation of 3 new training triples with their probabilities of occurrence

As shown, there are four ground truth entities (*Amsterdam, Netherlands, Rotterdam, and Europort*) with their three ground truth types of relations (*capital_of*, *located_in*, and *has_port*). Based on them, the recommender recommends relations that are most likely to occur - *birthplace_of* and *operates_in*. Then, our model adds the three new triples to the training set. The new entities (either head or tail) are assigned to the particular label. Thus, the maximum number of new entities matches the number of labels in the training set.

Creating new entities introduces a challenge, as they often have a high degree - many relations. To overcome this, we introduce **sampling** - for each newly added set of relations, we shuffle them and delete a fixed percentage (sampling rate, e.g., 30%) of them. Additionally, to improve computational efficiency during testing, we introduce two further **numeric thresholds** that limit the number of triples regardless of other factors. They halt the addition of new triples once a specified limit of triples or entities is reached.

Another threshold we introduce is the **maximum number of new relations** added to a given entity. It is possible that some entities have a limited number of edges, e.g., 1. Then, the recommender could recommend many possible edges. This would result in the loss of the original semantics of the KG. To mitigate this, a maximum threshold is applied: for each entity y with x original relations, the number of added relations to the entity y is limited to x.

For research purposes, we introduce other solutions for adding new training triples. One of them adds **only outgoing recommendations** and therefore uses a different SchemaTree. We also implement a *typed* process, where we add **an additional prefix** to each property representing the type of the entity it has. For example, a typed incoming property *prop* with a type *type* has the following structure: *I:typeprop*. Because the type of FB15k-237 often has many types of one entity, we create a list of all types and sort them from the most popular to the least. Then, for each entity's set of types, we choose only the most frequent one in the dataset. We exclude the most frequent type, */common/topic*, due to its overrepresentation (14,939 occurrences vs. 4,507 for the next most common).

3.3 Leave-one-out weighted training

This model introduces another approach where we manipulate the embedding loss function. This enhances the embedding training by assigning differential weights during the loss calculation. Those dynamic weights are based on the likelihood of the relations.

The fundamental principle employs the **Leave-One-Out** (LOO) mechanism - the calculation of individual triple likelihood to be involved in actual queries. Consider the following example. We want to calculate the weight score of a triple *Obama -> O:lives_in -> Washington*. Firstly, we pass the list of all properties of the entity *Obama* to the Recommender System. However, we exclude the target outgoing property *O:lives_in* since the recommender does not return properties involved in the query. Then, the recommender returns a list of possible properties with their probabilities. If one of them is our target, *O:lives_in*, we take its probability. Then, we do the same process but for the tail entity, *Washington*

- we query the recommender with a set of all *Washington*'s properties without the target *I:lives_in*, which in this case is an incoming property. Then, once the recommender returns a set of labels with the target *I:lives_in*, we take its probability. Now that we have 2 probabilities of *lives_in*, we take an average of them to obtain a balanced assessment that considers the triple's query relevance from both viewpoints. We denote this outcome as the **LOO probability score**. If the recommender does not return the property at all, then the score is 0 - neutral.

The second step in the LOO approach is to integrate the LOO probability scores into the training process through a custom weighted training loop that modifies PyKEEN's standard SLCWA (the stochastic local closed world assumption [4]). Here, for each training triple, we assign a weight based on the LOO probability score. Weights are scaled by a configurable multiplier (e.g., 5.0) to amplify the differences between high and low-scoring triples. During each training batch, the loss function is weighted according to the individual triple LOO scores, emphasizing more query-relevant triples.

3.4 Statistical tests

We run the extended ComplEx with hyperparameters that used Ruffinelli, D. et al. (2020) [33] with the same dataset. The results are compared with the baseline ComplEx model that we run on the same hyperparameters but with the original training set and weights. At the end, we save the ranks of both the extended and baseline models and perform Welch's t-tests on *MRR* and *Hits@K*. The P value is set as 0.05. The statistical test implementation and visualization are inspired by Daniel D. et al. (2023) [14].

4 Results

In this section, we present the results of our two extended models along with their analysis. Throughout the project, we adhered to scientific methodology, set a fixed random seed, and applied statistical evaluation methods described in the Methodology section. While comparing models, the embedding hyperparameters were the same. As a result, all experiments are reproducible, and the code is available in the accompanying GitHub repository [27]. The run time is shown in the table 8 in the appendix. While testing the full models, we noticed that the baseline ComplEx did not reproduce the Ruffinelli, D. et al (2020) [33] results - our accuracy was lower.

4.1 Bidirectional artificial triples

For each of the models, we ran a grid hyperparameter search for the specific triples extension methods in order to find the most optimal parameters, such as the probability threshold and sampling. Hence, the presented results are from models trained with hyperparameters that were found to be the most optimal.

We also performed several experiments with different setups. They include the addition of only outgoing relations as well as both incoming and outgoing (bidirectional). We also included the types of entities. We divided the experiment results into two phases: an early stage (first 19 epochs to analyze the early convergence) and a final phase after full training. The former results are presented in Table 1.

Model	Hits@1	Hits@3	Hits@10	MR	MRR	New triples
Baseline ComplEx	0.037	0.085	0.194	572.4	0.089	0
Outgoing Pr: 0.25	0.053	0.112	0.239	638.0	0.113	58595 (21.53%)
Bidirectional Pr: 0.25	0.056	0.120	0.247	622.5	0.118	81345 (29.89%)
Bidirectional-typed Pr: 0.25	0.046	0.120	0.243	651.9	0.109	63602 (23.37%)

Table 1. Additional triples models comparison of early training stages (Complex, for both head and tail, realistic, 19 epochs, Sa: 0.0)

In the Table 1:

- *New triples*: Triples added to the test set based on the recommendations (without any addition, in the training set there are 272,115 triples) and what percentage of the original number it is
- *MR*: Arithmetic Mean Rank
- *MRR*: Mean Reciprocal Rank (Inverse Harmonic Mean Rank)
- *Outgoing*: Trained on only outgoing relations
- *Bidirectional*: Trained on both incoming and outgoing relations
- *Prob*: Probability threshold that limits the number of new triples
- *Sa*: Sampling rate (e.g., 0.1-random 10 % of new added triples deleted)

Welch's t-test confirms that the improvement over the baseline model in terms of the first epochs is statistically significant for all extended models. A visual comparison is shown in Fig. 8 in the appendix. A learning curve of the baseline and bidirectional is presented in Fig. 5.

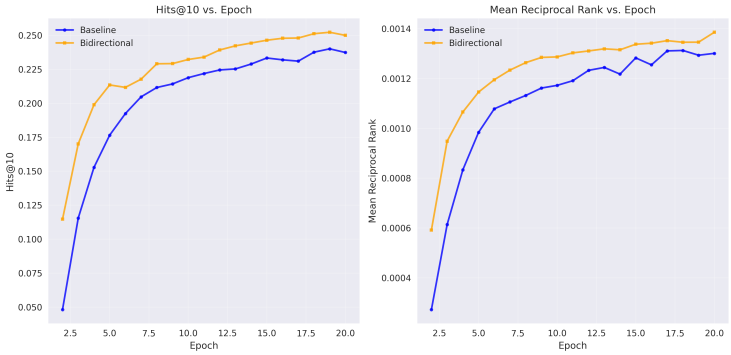

Fig. 5. Models' learning curve comparison.

The extended model outperforms the baseline ComplEx in the early stages of the training, including the initial epochs. In the appendix, we included the

Hits@1 and *Hits@3* learning curves in Fig. 9. The results indicate that the Recommender System achieves significantly better performance than the baseline within the first few epochs, up to 50% for *Hits@1* and 26% for *Hits@10*. However, the faster convergence in terms of training steps comes with increased per-epoch computational cost. Nevertheless, the training time for the extended model is only 18% seconds longer with a 0.5 probability threshold.

The results from the final stages of the training indicate that once the embedding models are trained for 100 epochs or more, the extended model does not outperform the baseline model (see table 7 in the appendix).

4.2 Leave-one-out weighted training

This method was also examined on the early and later stages separately. Similarly to the first method, we present the results of training after 19 epochs and the final (more than 100 epochs), and compare them with the baseline ComplEx trained using the same hyperparameters.

The results indicate that the LOO method decreases the performance in the early stages of the ComplEx training (see the table in the appendix 5). Once the model training converges, the results of both the LOO and baseline ComplEx models are statistically indistinguishable, indicating that they achieve similarly optimal performance. We also performed a *hybrid* - the extension of the training set (the bidirectional model) and the LOO weighting method. The results indicate that while the hybrid model converges in fewer epochs, the final metric scores are not statistically significantly improved (see the table in the appendix 6). The training time is the same as the baseline.

4.3 Discussion

Extending the training set improves early-phase performance in terms of training steps across all metrics, likely due to the artificial triples that are a form of data augmentation that helps structure the embedding space early during training. This incurs only a modest increase in training time. The new triples, even though they are not the ground truth, preserve the semantics of the original KG. This answers our research question that prior knowledge of likely-to-be-queried edges improves the training, but only by early acceleration in terms of training steps. Our findings suggest that while the proposed approach benefits the model by achieving better performance with reduced training iterations, both models after the full training can converge to comparable final performance when training runtime is not a constraint.

Interestingly, when extending the model using only outgoing relations, the precision did not significantly decrease compared to the bidirectional variant. This might indicate that extending the model on both incoming and outgoing relations might be redundant when working with limited computational power. Additionally, including entity type information did not yield performance improvements, indicating that the structural patterns captured by relation triples alone may already encode sufficient semantic context for the extended method.

However, in the case of the second approach - manipulating the weights by the LOO method, the model does not outperform the baseline model, also in the first phases of the training. We hypothesize that the LOO-based weighting scheme introduces a bias toward triples deemed query-relevant by the recommender, potentially limiting the model's exposure to less frequent but semantically valuable patterns. Also, the weighting might distort the gradient landscape, leading to unstable or even suboptimal updates, particularly when low-scoring triples are downweighted excessively. These findings suggest that incorporating the relevance-aware signals based on the label recommendations can be detrimental if not carefully calibrated, as it may decrease performance on unseen data.

4.4 Conclusion and future work

The experiments show that integrating prior information about likely queried labels purely on the training set can statistically significantly improve the initial stages (epochs) of the training across all measured metrics. This is, however, not true for manipulating the loss function. Additionally, in both cases, the baseline model eventually matches the performance of the pre-informed trained variants during later stages of training. The lack of improvement of the final model might be due to the recommender architecture. The recommender's ability to return only the labels that are not part of the query might limit our pre-informed training approaches. Also, creating possible positives and manipulating the weights might be overcomplicating and overinforming the embedding model. Promising results based on a limited number of epochs might be due to the fact that the baseline model is simply not yet trained enough to generate reasonable outcomes, and, accordingly, most of the new artificial triples will benefit the model.

Although the limited impact on final accuracy, extending the training set with recommender-informed triples improves early-stage convergence in terms of optimization steps. Incorporating this approach into Graph Neural Networks (GNNs) might retain early benefits and enhance overall performance. Especially, using the *ULTRA* model presented by Mikhail G. et al. (2024) [18] could gain from using the pre-informed trained methods. This extended relational representation may enhance the model's ability to generalize and better capture the underlying graph structure. Additionally, extending the Recommender System to include labels present in the query could potentially further improve accuracy. On the other hand, the LOO method needs further investigation, possibly by extensions of the weighting mechanism. The generalizability to larger graphs or other real-world systems remains to be validated. To conclude, further work is needed in the pre-informed link prediction tasks.

References

1. Abedjan, Z., Naumann, F.: Improving rdf data through association rule mining. Datenbank-Spektrum **13**, 111–120 (2013)
2. Abedjan, Z., Naumann, F.: Amending rdf entities with new facts. In: The Semantic Web: ESWC 2014 Satellite Events: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers 11. pp. 131–143. Springer (2014)
3. Abu-Salih, B.: Domain-specific knowledge graphs: A survey. Journal of Network and Computer Applications **185**, 103076 (2021)
4. Ali, M., Berrendorf, M., Hoyt, C.T., Vermue, L., Galkin, M., Sharifzadeh, S., Fischer, A., Tresp, V., Lehmann, J.: 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), 8825–8845 (2021)
5. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., Vrgoč, D.: Foundations of modern query languages for graph databases. ACM Computing Surveys (CSUR) **50**(5), 1–40 (2017)
6. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys (CSUR) **40**(1), 1–39 (2008)
7. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: international semantic web conference. pp. 722–735. Springer (2007)
8. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An introduction to description logic. Cambridge University Press (2017)
9. Balazevic, I., Allen, C., Hospedales, T.: Multi-relational poincaré graph embeddings. Advances in neural information processing systems **32** (2019)
10. Bernardi, A., Costabello, L.: Domain and range aware synthetic negatives generation for knowledge graph embedding models. arXiv preprint arXiv:2411.14858 (2024)
11. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1247–1250 (2008)
12. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. Advances in neural information processing systems **26** (2013)
13. Cochez, M.: Recommenderserver. `https://github.com/martaannaj/RecommenderServer/` (2024), accessed: 2025-03-20
14. Daza, D., Alivanistos, D., Mitra, P., Pijnenburg, T., Cochez, M., Groth, P.: Bioblp: a modular framework for learning on multimodal biomedical knowledge graphs. Journal of Biomedical Semantics **14**(1), 20 (2023)
15. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32 (2018)
16. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 601–610 (2014)
17. Ehrlinger, L., Wöß, W.: Towards a definition of knowledge graphs. SEMANTiCS (Posters, Demos, SuCCESS) **48**(1-4), 2 (2016)

18. Galkin, M., Yuan, X., Mostafa, H., Tang, J., Zhu, Z.: Towards foundation models for knowledge graph reasoning. arXiv preprint arXiv:2310.04562 (2023)
19. Gleim, L.C., Schimassek, R., Hüser, D., Peters, M., Krämer, C., Cochez, M., Decker, S.: Schematree: Maximum-likelihood property recommendation for wikidata. In: European Semantic Web Conference. pp. 179–195. Springer (2020)
20. Gul, H., Bhat, A.A., Naim, A.G.H.: Muco-kgc: Multi-context-aware knowledge graph completion. arXiv preprint arXiv:2503.03091 (2025)
21. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G.D., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., et al.: Knowledge graphs. ACM Computing Surveys (Csur) **54**(4), 1–37 (2021)
22. Hussein, R., Yang, D., Cudré-Mauroux, P.: Are meta-paths necessary? revisiting heterogeneous graph embeddings. In: Proceedings of the 27th ACM international conference on information and knowledge management. pp. 437–446 (2018)
23. Jain, N., Tran, T.K., Gad-Elrab, M.H., Stepanova, D.: Improving knowledge graph embeddings with ontological reasoning. In: International Semantic Web Conference. pp. 410–426. Springer (2021)
24. Ji, S., Pan, S., Cambria, E., Marttinen, P., Yu, P.S.: A survey on knowledge graphs: Representation, acquisition, and applications. IEEE transactions on neural networks and learning systems **33**(2), 494–514 (2021)
25. Kumar, A., Singh, S.S., Singh, K., Biswas, B.: Link prediction techniques, applications, and performance: A survey. Physica A: Statistical Mechanics and its Applications **553**, 124289 (2020)
26. Liu, Y., Tian, X., Sun, Z., Hu, W.: Finetuning generative large language models with discrimination instructions for knowledge graph completion. In: International Semantic Web Conference. pp. 199–217. Springer (2024)
27. Meczynski, S.A.: Pre-informed-training. `https://github.com/slawmecz/link-prediction-thesis` (2024), accessed: 2025-04-01
28. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016)
29. Nurdiati, S.S., Hoede, C.: 25 years development of knowledge graph theory: the results and the challenge (2008)
30. Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15. pp. 498–514. Springer (2016)
31. Rossi, A., Barbosa, D., Firmani, D., Matinata, A., Merialdo, P.: Knowledge graph embedding for link prediction: A comparative analysis. ACM Transactions on Knowledge Discovery from Data (TKDD) **15**(2), 1–49 (2021)
32. von Rueden, L., Houben, S., Cvejoski, K., Bauckhage, C., Piatkowski, N.: Informed pre-training on prior knowledge. arXiv preprint arXiv:2205.11433 (2022)
33. Ruffinelli, D., Broscheit, S., Gemulla, R.: You can teach an old dog new tricks! on training knowledge graph embeddings (2020)
34. Schneider, E.W.: Course modularization applied: The interface system and its implications for sequence control and data analysis. (1973)
35. Sigurbjörnsson, B., Van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: Proceedings of the 17th international conference on World Wide Web. pp. 327–336 (2008)
36. Singhal, A., et al.: Introducing the knowledge graph: things, not strings. Official google blog **5**(16), 3 (2012)
37. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web. pp. 697–706 (2007)

38. SURF: Snellius: De nationale supercomputer. `https://www.surf.nl/diensten/rekenen/snellius-de-nationale-supercomputer` (2024), accessed: 2025-04-20
39. Toutanova, K., Chen, D.: Observed versus latent features for knowledge base and text inference. In: Proceedings of the 3rd workshop on continuous vector space models and their compositionality. pp. 57–66 (2015)
40. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: International conference on machine learning. pp. 2071–2080. PMLR (2016)
41. Voorhees, E.M., et al.: The trec-8 question answering track report. In: Trec. vol. 99, pp. 77–82 (1999)
42. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE transactions on knowledge and data engineering **29**(12), 2724–2743 (2017)
43. West, R., Gabrilovich, E., Murphy, K., Sun, S., Gupta, R., Lin, D.: Knowledge base completion via search-based question answering. In: Proceedings of the 23rd international conference on World wide web. pp. 515–526 (2014)
44. Wood, D., Lanthaler, M., Cyganiak, R.: Rdf 1.1 concepts and abstract syntax. W3C Recommendation, W3C (2014)
45. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575 (2014)
46. Yao, L., Mao, C., Luo, Y.: Kg-bert: Bert for knowledge graph completion. arXiv preprint arXiv:1909.03193 (2019)
47. Zangerle, E., Gassler, W., Pichl, M., Steinhauser, S., Specht, G.: An empirical evaluation of property recommender systems for wikidata and collaborative knowledge bases. In: Proceedings of the 12th International Symposium on Open Collaboration. pp. 1–8 (2016)
48. Zhong, L., Wu, J., Li, Q., Peng, H., Wu, X.: A comprehensive survey on automatic knowledge graph construction. ACM Computing Surveys **56**(4), 1–62 (2023)
49. Zhu, Y., Wang, X., Chen, J., Qiao, S., Ou, Y., Yao, Y., Deng, S., Chen, H., Zhang, N.: Llms for knowledge graph construction and reasoning: Recent capabilities and future opportunities. World Wide Web **27**(5), 58 (2024)

Appendix

Hyperparameters influence

While experimenting on different hyperparameters, we noticed that the most influential one is the **probability threshold**. Fig. 6 illustrates how this hyperparameter influences the model.

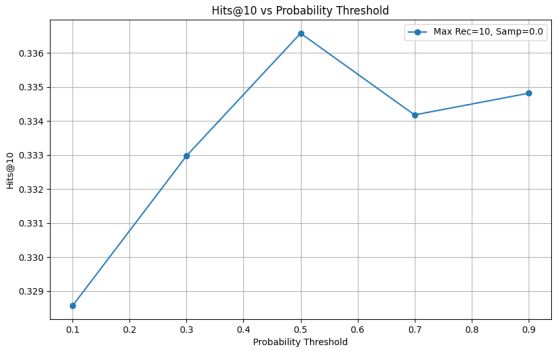

Fig. 6. ComplEx Bidirectional hyperparameter search visualisation - probability threshold vs Hits@10; 19 epochs, sampling 0.0

The differences between runs are minor; nevertheless, after running over this and other configurations, we noticed that higher probability thresholds - adding fewer but more accurate triples - consistently improve *Hits@10*, suggesting the outgoing model benefits from prioritizing high-confidence triples. However, once the probability threshold reaches 0.5, the accuracy slightly decreases. This results from limited accuracy gains at a 0.6 probability threshold (shown in Fig. 3 and explained in the 3.1 subsection), with further declines at higher values.

However, the highest F1 score for recommended triples occurs at a 0.25 threshold (Figure 3), indicating that peak F1 does not necessarily align with optimal *Hits@10*. This pattern does not hold for *Hits@3* and *Hits@1*. A bar chart on Fig. 7 illustrates the most effective thresholds for each metric.

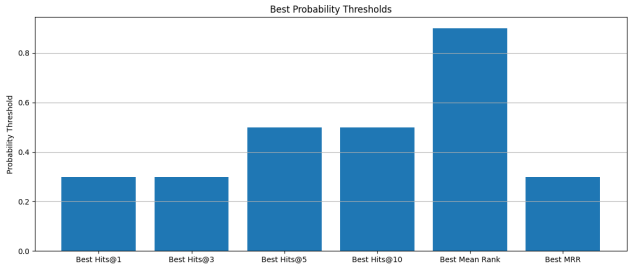

Fig. 7. Optimal Probability Thresholds for Each Evaluation Metric for ComplEx Bidirectional

The lower k in the *Hits@k* metrics, the lower the probability threshold works better. The reason might be that lower probability thresholds result in a greater

number of training triples, including those with lower confidence scores. This increased coverage allows the model to learn a broader range of relational patterns, which can improve its ability to rank the correct entity at top positions - *Hits@1* and *Hits@3*. In contrast, higher probability thresholds filter the training data to include only the most confidently predicted triples. While this may limit the diversity of the training signal, it reduces noise. As a result, higher thresholds tend to be more beneficial for metrics like *Hits@10*, where general accuracy over a broader candidate set is more important than precise top-rank performance.

On the other hand, increasing the **sampling rate** - whereby a portion of the newly added triples is discarded - has a similar effect on the model as increasing the probability threshold. For instance, if raising the probability threshold results in the exclusion of 10% of candidate triples, this resembles applying a 10% sampling rate. However, the statistical analysis results show that probability thresholding tends to be more effective in improving the MRR metrics. This might be because the probability thresholding systematically filters out less confident triples based on model uncertainty, whereas random sampling removes information indiscriminately (randomly). This suggests that simple sampling is a less efficient strategy for controlling triple quality.

The functionality that limits the maximum number of new properties of an entity X by the original number of entities of the entity X does not change the results statistically. This is because, in most cases, the Recommender System does not recommend significantly more properties with a high probability of occurrence than the entity already has.

The additional hyperparameters - the overall number of new properties and triples - were not used in the final results, as they were made for testing purposes to limit the computational power and time of testing.

Model	**Hits@1**	**Hits@3**	**Hits@10**	**MRR**
ComplEx	0.253	0.384	0.536	0.348

Table 2. Performance of ComplEx on FB15k-237, Ruffinelli, D. et al. (2020).

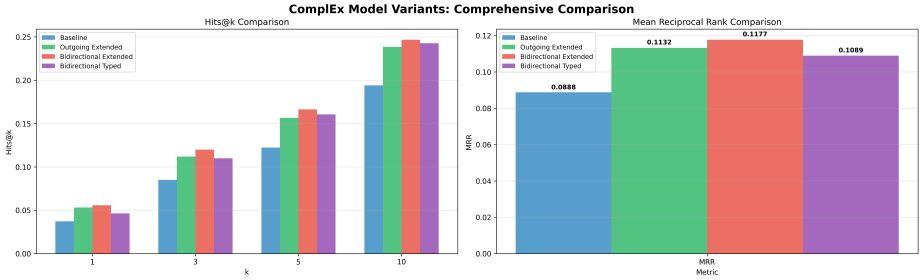

Fig. 8. Comprehensive extended models comparison.

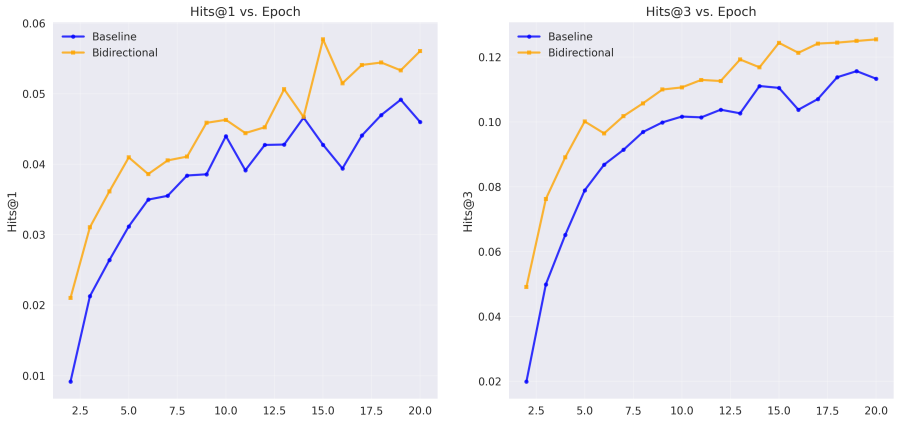

Fig. 9. Training metrics comparison, Sampling rate 0.0, Probability threshold 0.25

Dataset	Relations	Entities	Train	Validation	Test
FB15K	1,345	14,951	483,142	50,000	59,071
FB15KSelected	237	14,541	272,115	17,535	20,466

Table 3. Statistics of FB15K and FB15K-237 (FB15KSelected) datasets.

Property Range	**Description**	**Number of Entities**
1–5	Very few	4682
6–10	Few to moderate	3158
11–15	Moderate	3192
16–20	Moderately high	2377
21–30	High	1208
31–40	Very high	88
41–50	Extremely high	58
51+	Outliers	3

Table 4. Distribution of Entities by Number of Total Properties (Grouped)

Model	Hits@1	Hits@3	Hits@10	MRR
Baseline ComplEx	0.037	0.085	0.194	0.089
LOO - weighted training	0.025	0.064	0.156	0.068

Table 5. Leave-one-out weighted training results after 19 epochs

Model	Hits@1	Hits@3	Hits@10	MRR	New triples
Baseline ComplEx	0.183	0.291	0.417	0.263	0
LOO - weighted training	0.180	0.286	0.410	0.258	0
LOO + Bidirectional	0.181	0.285	0.400	0.256	95,474 (35.09%)

Table 6. Leave-one-out weighted training results after 100 epochs, Sampling rate 0.0, Probability threshold 0.25

Model	Hits@1	Hits@3	Hits@10	MR	MRR	New triples
Baseline ComplEx	0.183	0.291	0.417	415.7	0.263	0
Bidirectional Pr: 0.5	0.180	0.282	0.398	564.3	0.255	50744 (18,64%)

Table 7. Final additional triples models comparison (Complex, for both head and tail, realistic, 100 epochs, Sampling rate 0.0, Probability Threshold 0.5)

Model	Added triples	Training time (in minutes)
Baseline ComplEx	0	81
Bidirectional Pr: 0.5	50,744 (+19%)	96
Weighted:	0	82

Table 8. Comparison between training time on Snellius with gpu a_100, 19 epochs