KGxBoard: Explainable and Interactive Leaderboard for Evaluation of Knowledge Graph Completion Models

Haris Widjaja*[1], **Kiril Gashteovski***[2], **Wiem Ben Rim**[2], **Pengfei Liu**[1,4], **Christopher Malon**[3], **Daniel Ruffinelli**[5], **Carolin Lawrence**[2], **Graham Neubig**[1,4]
[1] Carnegie Mellon University; [2] NEC Laboratories Europe;
[3] NEC Laboratories America; [4] Inspired Cognition; [5] University of Mannheim;

Abstract

Knowledge Graphs (KGs) store information in the form of *(head, predicate, tail)*-triples. To augment KGs with new knowledge, researchers proposed models for KG Completion (KGC) tasks such as link prediction; i.e., answering *(h; p; ?)* or *(?; p; t)* queries. Such models are usually evaluated with averaged metrics on a held-out test set. While useful for tracking progress, averaged single-score metrics cannot reveal *what exactly* a model has learned—or failed to learn. To address this issue, we propose KGxBoard[1]: an interactive framework for performing fine-grained evaluation on meaningful subsets of the data, each of which tests individual and interpretable capabilities of a KGC model. In our experiments, we highlight the findings that we discovered with the use of KGxBoard, which would have been impossible to detect with standard averaged single-score metrics.

1 Introduction

Knowledge Graphs (KGs) are graph databases that store information about entities and the relations between them in the form of *(head, predicate, tail)*-triples (Weikum et al., 2021). Because of their flexible structure, KGs are used for storing general real-world data (Rebele et al., 2016) as well as domain-specific data covering various domains (Abu-Salih, 2021), including medicine (Chandak et al., 2022), IoT (Le-Phuoc et al., 2016) and finance (Cheng et al., 2020). KGs play an important role in NLP: they are used for many downstream tasks, including language modeling (Logan et al., 2019), entity linking (Radhakrishnan et al., 2018) and question answering (Saxena et al., 2022).

A common problem for KGs is that they are incomplete (i.e., they do not contain all facts about the world or the particular domain), which could lead to limitations in performance for the downstream tasks. To tackle this problem of incompleteness, the research community has worked on many KG Completion (KGC) tasks, most prominently on *link prediction*: predicting new facts within the KG by providing ranked predictions to the queries *(h; p; ?)* or *(?; p; t)* (Clouatre et al., 2021). Models for KGC tasks are typically evaluated with single-score metrics that are averaged over a held-out test set. For instance, for a KGC query—e.g., *(h; p; ?)*—, hits@k indicates the average number of correct answers from the test set that appear within the top-k ranked entities predicted by the KGC model.[2]

While using such scores is important for tracking the progress of KGC models, researchers observed that a more fine-grained evaluation is needed (Palmonari and Minervini, 2020), because the averaged metrics cannot answer the question of *what properties* have the models actually learned—or failed to learn. To investigate what properties were learned by the KGC models, researchers have designed specific datasets and experimental setups. For example, Rim et al. (2021) used the idea of behavioral testing applied to NLP models (Ribeiro et al., 2020) to perform more fine-grained tests for relation symmetry. In particular, they measured the performance of KGC models for queries that contain symmetric relations; i.e., relations that are true for both *(h; p; t)* and *(t; p; h)* such as *(X; marriedTo; Y)*. However, their proposed framework only contains a limited number of tests and might not cover model properties of interest to other researchers.

To make the fine-grained evaluation more generic—and to compare different KGC models across different properties—we propose KGxBoard: a method and software implementation for fine-grained evaluation of KGC models (see illustration in Fig. 1). KGxBoard splits the evaluated data into groups (buckets) according to certain properties of the data. For instance, one can

*Equal contribution

[1]All resources (code, data, demo, video, etc.) are available on `https://github.com/neulab/KGxBoard`

[2]See Sec. 2 for more details about hits@k and other metrics

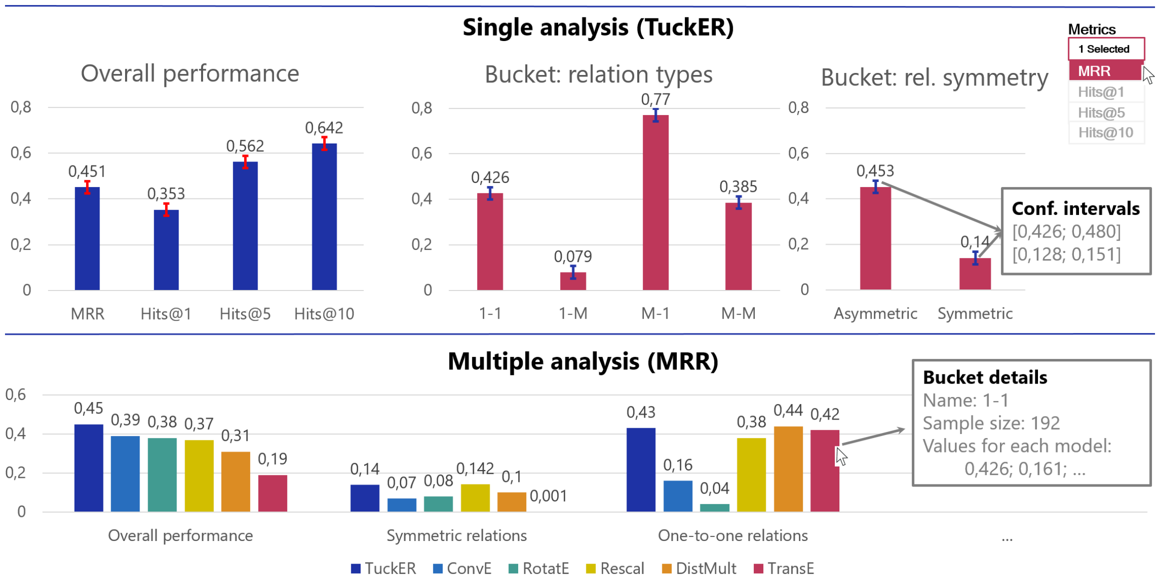


Figure 1: Illustration of KGxBoard's functionalities. With the *single analysis mode*, the user can see the overall performance of a given KGC model, as well as the performance of the model across multiple buckets (i.e., partitions of the evaluated data). For example, we observed that the TuckER model performs very well on triples with Many-to-one (M-1) relations. However, it performs poorly on triples with One-to-many (1-M) relations and symmetric relations. With the *multiple analysis mode*, the user can compare the overall and bucketized performance across different models. Such a view enables the user to compare the models in a fine-grained manner; e.g., Rescal is ranked 4th in the overall performance, but it is ranked 1st for the triples having symmetric relations. Likewise, DistMult is ranked 5th in the overall performance and 1st for the triples with One-to-one relations. In both modes, the user can interact with the UI to see details of the data buckets, change the evaluation metrics, etc.

split the test data into two buckets of data points such that one bucket contains triples with symmetric and the other with asymmetric relations. Consequently, the users can observe the performance of each of these buckets separately, while also having an overview of the overall performance scores.

KGxBoard builds upon ExplainaBoard (Liu et al., 2021), which is an explainable leaderboard for NLP tasks. We adapt ExplainaBoard to the KGC tasks[3] by providing: (1) a method and software implementation for fine-grained evaluation of KGC models, integrated into ExplainaBoard; (2) APIs for porting results from two popular KGC frameworks—PyKeen (Ali et al., 2021) and LibKGE (Broscheit et al., 2020)—directly into KGxBoard format; (3) interface for reading customized features for fine-grained evaluation; (4) experimental study exposing problems with KGC models that cannot be spotted with averaged scoring; (5) experimental study showing that the findings from the fine-grained evaluation of KGxBoard can be used for automatic debugging of the models.

[3]KGxBoard can handle the KGC tasks on answering queries of the form *(?, p, t)*, *(h, ?, t)* and *(h, p, ?)*. For simplicity, when we refer to KGC models in this paper, we refer to models that provide predictions of the form *(h, p, ?)*.

2 Preliminaries

The performance of KGC models is evaluated on a held-out test set of golden KG triples. In particular, the models return a set of ranked answers to the queries—e.g., *(h; p; ?)*—, where the correctness of the answers is evaluated against the golden triples. The final score is averaged over the examples from the held-out test set. The standard scores used in the literature are:

(1) Hits@$k = \frac{1}{|\mathcal{D}|} \sum_{(h,r,t)\in\mathcal{D}} \mathbb{1}[\texttt{rank}(t) \leq k]$

(2) MRR $= \frac{1}{|\mathcal{D}|} \sum_{(h,r,t)\in\mathcal{D}} \frac{1}{\texttt{rank}(t)}$

(3) MR $= \frac{1}{|\mathcal{D}|} \sum_{(h,r,t)\in\mathcal{D}} \texttt{rank}(t)$

where $\mathcal{D}$ is the set of test triples, and h, r, t are the head, relation and tail of the KG triple respectively.

3 KGxBoard's Fine-grained Evaluation

Prior work in NLP has pointed to the issues of using single-score metrics, which do not expose what exact properties of the data were (or were not) learned by the models (Narayan et al., 2021); e.g., some information extraction models perform poorly when there is a conjunction present in the

```
"custom_features": {
    "rel_type": {
        "dtype": "string",
        "description": "predicate symmetry",
        "num_buckets": 2
    }
}
```

Figure 2: Defining custom features for bucketization of the evaluation data (e.g., validation or test data).

sentence (Gashteovski et al., 2022). To better understand what properties were (or were not) learned by the NLP models, people have proposed multi-faceted or explainable and interactive leaderboards (for more details on related work, see Appendix A).

Following ExplainaBoard (Liu et al., 2021), the basic idea of KGxBoard's fine-grained evaluation is to breakdown the performance measure (e.g., Hits@10) over individual groups (buckets) in addition to the performance score over the overall evaluation dataset. This approach involves three steps: (i) define features upon which the evaluation dataset is going to be partitioned; (ii) partition evaluation dataset into different buckets based on the defined features; and (iii) calculate performance w.r.t. each bucket. In contrast to ExplainaBoard, KGxBoard is tailored for the KG completion tasks and their evaluation metrics.

3.1 Feature definition

The feature definition describes the manner upon which KGxBoard is going to partition the evaluated data into buckets. For example, if the feature is about predicate symmetry, then the data is divided in 2 buckets: (1) triples that have predicates which are considered symmetric;[4] (2) triples with asymmetric predicates. Because each data point can be assigned to one of these buckets with a label *"symmetric"* or *"asymmetric"* (strings), the user defines the feature "predicate symmetry" accordingly (as shown on Figure 2).

Built-in Features. KGxBoard supports several built-in features that will automatically bucketize the evaluation data of any models to be analyzed. The built-in features include several widely used properties of the data, such as predicate symmetry (Trouillon et al., 2016) and entity type hierarchy (Rim et al., 2021); see details in Appendix B.

[4]predicates that are true for both *(h; p; t)* and *(t; p; h)*; e.g., the predicate *p=marriedTo*: *(x; p; y)* $\Longleftrightarrow$ *(y; p; x)*.

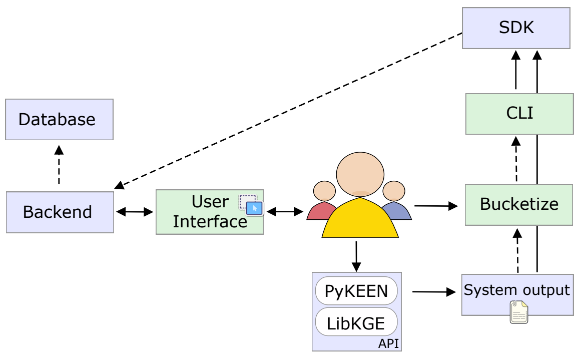


Figure 3: General overview of KGxBoard's architecture. SDK: Software Development Kit; CLI: Command-Line interface.

Customized Features. KGxBoard also allows users to customly define their own features by specifying additional information in the system output file. If, for example, the users want to define the bucketization features for predicate symmetry, then they only need to specify this in a json file (Fig. 2).

3.2 Partitioning of Evaluation Data into Different Buckets

For the built-in features, KGxBoard automatically assigns each data point to its respective bucket. For the customized features, once the custom features were defined, the user should place each data point to its respective customized bucket via the bucketization functions (explained in Section 4.4). This data is then fed into KGxBoard, which automatically computes the relevant metrics for each bucket.

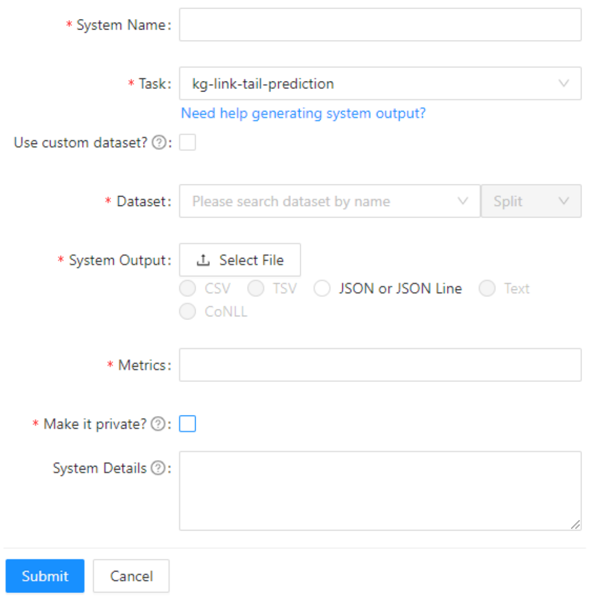
3.3 Calculate Performance w.r.t. each Bucket

KGxBoard computes the relevant metrics (described in Section 2) for each bucket individually, as well as for the overall evaluation dataset.

Confidence Interval. KGxBoard has been endowed with the ability to quantify to what degree we can trust the result of each bucket. Specifically, as illustrated in Figure 1, each bin has been equipped with an error bar and its width reflects the reliability of the bucket performance. KGxBoard supports two ways to calculate the confidence interval: bootstrapped re-sampling (Efron, 1992) and t-test (Nakagawa and Cuthill, 2007) .

4 KGxBoard: System Overview

A general overview of the KGxBoard architecture is illustrated in Figure 3. The users can provide the data for the models through the front-end (via the


* System Name:

* Task: kg-link-tail-prediction

Need help generating system output?

Use custom dataset? ☐

* Dataset: Please search dataset by name | Split

* System Output: Select File

CSV | TSV | JSON or JSON Line | Text | CoNLL

* Metrics:

* Make it private? ☐

System Details

Figure 4: Using the frontend to upload a new model.

UI) or the back-end (via the CLI or the SDK). Then, these results are stored in a database (DB), which can be accessed and viewed either with the visual interface, programmatically, or with the command-line interface. In general, the users can choose three ways to use KGxBoard's functionalities: (i) directly from the interactive web interface; (ii) through an API from KGC frameworks (PyKEEN and LibKGE); (iii) through the command-line interface with already provided data.

4.1 Frontend

We adopt a React-based technology stack[5] to create an interactive web app as the frontend of KGxBoard. Assuming that a user has generated the input data in prior steps (e.g., through the API from KGC libraries; see Section 4.3), she can upload the data via the frontend interface (Figure 4). The data is then passed on to the backend, which stores it in a database. The frontend provides two types of analysis of the models' evaluation: **single analysis** aims to identify the strengths and weaknesses of a given KG completion model; **pairwise/multiple analysis** can help users figure out where one model is better (or worse) than the other when two (or more) models are selected; for illustration of the frontend, see Figure 1.

4.2 Backend

The backend is built on top of ExplainaBoard's backend code (Liu et al., 2021). The main functionalities for KGxBoard's backend are: (i) defining the evaluation metrics for the link prediction task: Hits@k, MRR and MR; (ii) computing the overall scores and the buckets' scores with the built-in features; (iii) handling customized feature buckets if the user provided any; (iv) storing the results in the DB; (v) communicate the results with the frontend in an interactive manner.

4.3 API with KGC Libraries

KGxBoard comes with APIs that can translate the output of two widely used KGC libraries—PyKEEN (Ali et al., 2021) and LibKGE (Broscheit et al., 2020)—into KGxBoard format. In particular, the APIs write the system-output files in KGxBoard format which does not contain buckets, only results from the models. If the user wishes to add customized buckets, she can either write or reuse already existing bucketization function(s), which will rewrite the data in KGxBoard format with the desired bucketizations.

4.4 Bucketization functions

The bucketization functions are procedures that do two main actions: (1) define the buckets; (2) assign a bucket label to each data example. Here's an example of a bucketization function pseudocode that defines relations as being either symmetric or asymmetric:

```
# s_rels -> set of symmetric rels.
def bucketize_rel_sym(s_rels):
    # define the buckets properties
    bucket_name = "rel_sym"; dtype = "string"
    descr="rel's symmetry prop."; num_buckets=2

    # assign bucket to example data
    for triple in predict_data:
        if triple['predicate'] in s_rels:
            triple.bucket = 'symmetric'
        else:
            triple.bucket = 'asymmetric'
```

5 Experimental Study

To showcase the usefulness of KGxBoard, we conducted the following experimental study: (1) bucketized comparison of models: we provide insights on the buckets' performance about different models trained with one KGC framework, which would have been impossible to discover with standard metrics; (2) comparison of models trained on different KGC frameworks with different hyperparameter settings: showing how KGxBoard can be used to discover differences between models trained in

[5]https://reactjs.org/

different environments; (3) automatic debugging: showing the ability of automatic debugging of the models by using insights from certain buckets.

5.1 Experimental Setup

Datasets. We used two widely used datasets: (1) FB15K-237 (Toutanova and Chen, 2015), constructed from Freebase (Bollacker et al., 2008), covering relations between people, locations, etc.; (2) WN18RR (Dettmers et al., 2018), constructed from WordNet (Miller, 1992) and represents connections between words in English, such as synonyms and hypernyms. The models were trained, validated and tested with the standard dataset split (the reported results are on the test sets).

Models and KGC Frameworks. For our experiments, we trained the KGC models on two commonly used KGC frameworks: PyKEEN and LibKGE. With PyKEEN we trained the following models for link prediction: DistMult (Yang et al., 2015), ConvE (Dettmers et al., 2018), RESCAL (Nickel et al., 2011), RotatE (Sun et al., 2019), TransE (Bordes et al., 2013) and TuckER (Balazevic et al., 2019). PyKEEN provides hyperparameters to reproduce the results of the original work of a given model, which we used to train the models. To compare the same models trained with different hyperparameters and frameworks, we also used the pretrained models obtained by Ruffinelli et al. (2020) as a result of extensive hyperparameter optimization using LibKGE (Broscheit et al., 2020). Specifically, we used the models ConvE, DistMult, Rescal and TransE (see training details in App. D).

Bucketizations. We partitioned the test data into different interpretable groups based on either built-in or customized features (e.g., relation type). All together, for FB15K-237 / WN18RR we have 303 / 35 unique buckets respectively. The large difference in the number of buckets between the datasets is mainly due to the significantly higher number of relations in the FB15K-237 dataset (237 v.s. 11). We provide further details on the buckets in App. C.

5.2 Bucketized Comparison of Models

To get an overview of the fine-grained evaluation of the KGC models, we used the predictions from PyKEEN on the FB15K-237 and WN18RR datasets. By ranking different systems with the MRR metric in two ways: (1) based on their overall performance; and (2) based on bucket-wise performance; we obtain that, as shown in Table 1, the overall ranking of the models is significantly different than the ranking of the models for each individual bucket.

	Overall rank	$b^{=}$	$b^{\neq}$
TuckER	1 / 1	.601 / .543	.399 / .457
ConvE	2 / 3	.301 / .486	.699 / .514
RotatE	3 / 5	.257 / .686	.743 / .314
Rescal	4 / 2	.261 / .457	.739 / .543
DistMult	5 / 4	.541 / .400	.459 / .600
TransE	6 / 6	.868 / .800	.132 / .200

Table 1: Ranking of KGC models (trained with PyKEEN) according to the MRR score. 1 indicates best rank. $b^{=}/b^{\neq}$: the fraction of buckets where the ranking of a given model is equivalent/not equivalent compared to the overall rank of the model. Results are on the datasets FB15K-237 / WN18RR.

For example, TuckER is ranked as the best-performing model on both FB15K-237 and WN18RR. However, TuckER is not ranked as best performing model for approximately 40% and 46% of the buckets for FB15K-237 and WN18RR respectively. For instance, when taking a closer look at FB15K-237, we find that for the bucket featurized by symmetric relations, TuckER is ranked 2nd and Rescal is ranked 1st (on the overall test set, Rescal is ranked 4th) and for the One-to-one relations, TuckER is also ranked 2nd and DistMult is ranked 1st (on the overall test set, DistMult is ranked 5th); see Figure 1.

Such findings would be impossible to spot with standard averaged metrics over the entire test set, and are similar in spirit to previous results that show how alternative evaluation methods can expose differences in overall model performance (Wang et al., 2019; Rim et al., 2021). With KGxBoard, researchers can diagnose issues with any KGC model on customized properties of the data (i.e., customized bucketizations of the evaluation data).

5.3 Bucketized Comparison of Models Trained with Different Hyperparameters

Similarly as with the fine-grained comparison between different models (Sec. 5.2), KGxBoard can be used to compare one model that was trained on multiple sets of different hyperparameters (HPs) and implementations. For this purpose, we train each of the four KGC models (ConvE, Rescal, DistMult and TransE) with two sets of HPs on two KGC libraries—PyKEEN and LibKGE—and showcase the differences. Note that the implementation from different libraries can vary significantly w.r.t. the

HP search space and the degree of customization.[6]

On the one hand, we trained each model with the hyperparameters that aim to reproduce the work of the original papers (trained with PyKEEN; the HP combination was proposed in PyKEEN's documentation). On the other hand, we used model configurations resulting from a HP optimization pipeline that ensures improved overall results (trained with LibKGE; HP combination proposed by Ruffinelli et al. (2020)). We refer to each of the former models as ORIGHP-MODEL and to the latter as OPTIMHP-MODEL. While the OPTIMHP models always outperform their ORIGHP counterparts, we still observed many cases where the ranking of the models flipped on some buckets. For example, ORIGHP-CONVE performs better than OPTIMHP-CONVE for: (1) the FB15K-237 triples with relation between award ceremony and award winner; (2) FB15K-237 triples whose tail entity types are of type "musical work" (details in App. E).

5.4 Automatic Debugging of Models Using Buckets

Hits@1	ConvE	TuckER	RotatE	Rescal
	Debugging test			
Before debug.	.0000	.0000	.0000	.0000
Naive	.0625	.1875	.0465	.1642
In-danger	.1562	.2083	.0465	.2015
	Original test			
Before debug.	.2710	.3108	.2627	.2596
Naive	.2416	.3010	.2627	.2594
In-danger	.2574	.3004	.2627	.2594

Table 2: Debugging results for the relations with most symmetry violations (Hits@1).

Grouping related properties of the model into buckets offers not only the potential to diagnose problems, but also the potential to fix them by debugging. We show here how the debugging techniques of Malon et al. (2022) may be adapted to fix problems with KG embeddings, illustrating the idea with one particular bucket: relation symmetry.

Updating the relation embedding to improve symmetry for one relation will have no effect on other relations generally, so we debug only one relation r at a time. For debugging, we collect a sub-bucket consisting of triples (h, r, t) that violate symmetry in a stricter sense: the reverse triple (t, r, h) is in the training set and the trained model predicts h as the tail for $(t, r, ?)$ with rank one, but t has rank greater than one among the predictions for $(h, r, ?)$. Ten triples from the sub-bucket constitute the *debugging set*, which is used to learn better model parameters, and the remaining triples are held out to form the *debugging test set*.

We debug the relation with the most symmetry violations (in the above sense) for four models: ConvE, TuckER, RotatE and Rescal.[7] A naive method, which we call *intensive fine-tuning*, simply fine-tunes the model on the debugging set alone until all its triples are predicted at rank one. We freeze entity embeddings during intensive fine-tuning, because updating the entity embedding will not generalize to improve symmetry on any held-out entities. To monitor whether this learning comes at the expense of forgetting other triples, we evaluate performance on the original test set before and after debugging, in addition to the debugging test set.

We also adapt the proposed method of Malon et al. (2022) to KGE. In this method, we first run the intensive fine-tuning, then collect twenty triples at random from the training set, which were correctly predicted at rank one after the original rank-tuning, but whose rank fell after the intensive fine-tuning. We use these twenty examples together with the ten debugging examples in a second round of intensive fine-tuning, again starting with the parameters of the original model. The hope is to learn the debugging examples while anchoring the performance of triples that are "in danger" of being forgotten.

Table 2 shows debugging results for the four models. In all cases, naive debugging improves Hits@1 on the held-out test debugging examples, and in-danger debugging often yields a further improvement. The impact of debugging on Hits@1 of the original test set is less than 1% for all models except ConvE, which has interaction parameters that are applied to many relations. For ConvE, the in-danger method cuts this sacrifice in half. We provide more detailed analysis in Appendix F.

6 Conclusions

We presented KGxBoard: an interactive framework for fine-grained and interpretable evaluation on meaningful subsets of the data, each of which tests individual and interpretable capabilities of a KGC model. We highlighted insights that would be impossible to detect with standard leaderboards.

[6]For instance, while LibKGE allows the customized initialization of the embeddings through HPs, PyKEEN requires changing the source code. More details in App. D.

[7]The other two models, Distmult and TransE, did not have enough symmetry violations to fill a debugging set.

References

Bilal Abu-Salih. 2021. Domain-specific Knowledge Graphs: A Survey. *Journal of Network and Computer Applications*, 185:103076.

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

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The semantic web*, pages 722–735. Springer.

Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. 2019. TuckER: Tensor Factorization for Knowledge Graph Completion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processin (EMNLP)*, pages 5184–5193.

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

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-Relational Data. *Advances in Neural Information Processing Systems (NeurIPS)*.

Samuel R. Bowman and George E. Dahl. 2021. What Will it Take to Fix Benchmarking in Natural Language Understanding? In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, page 4843–4855.

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 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, pages 165–174.

Payal Chandak, Kexin Huang, and Marinka Zitnik. 2022. Building a Knowledge Graph to Enable Precision Medicine. *bioRxiv*.

Dawei Cheng, Fangzhou Yang, Xiaoyang Wang, Ying Zhang, and Liqing Zhang. 2020. Knowledge Graph-based Event Embedding Framework for Financial Quantitative Investments. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 2221–2230.

Louis Clouatre, Philippe Trempe, Amal Zouaq, and Sarath Chandar. 2021. MLMLM: Link Prediction with Mean Likelihood Masked Language Model. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL)*, pages 4321–4331.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1811–1818.

Bradley Efron. 1992. Bootstrap Methods: Another Look at the Jackknife. In *Breakthroughs in Statistics*, pages 569–593. Springer.

Kawin Ethayarajh and Dan Jurafsky. 2020. Utility is in the Eye of the User: A Critique of NLP Leaderboards. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4846–4853.

Niklas Friedrich, Kiril Gashteovski, Mingying Yu, Bhushan Kotnis, Caroline Lawrence, Mathias Niepert, and Goran Glavaš. 2022. AnnIE: An Annotation Platform for Constructing Complete Open Information Extraction Benchmark. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL): System Demonstrations*, page 44–60.

Aldo Gangemi, Andrea Giovanni Nuzzolese, Valentina Presutti, Francesco Draicchio, Alberto Musetti, and Paolo Ciancarini. 2012. Automatic Typing of DBpedia Entities. In *Proceedings of the International Semantc Web Conference (ISWC)*, pages 65–81.

Kiril Gashteovski, Mingying Yu, Bhushan Kotnis, Carolin Lawrence, Mathias Niepert, and Goran Glavaš. 2022. BenchIE: A Framework for Multi-Faceted Fact-Based Open Information Extraction Evaluation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4472–4490.

Filip Ilievski, Daniel Garijo, Hans Chalupsky, Naren Teja Divvala, Yixiang Yao, Craig Rogers, Rongpeng Li, Jun Liu, Amandeep Singh, Daniel Schwabe, et al. 2020. KGTK: A Toolkit for Large Knowledge Graph Manipulation and Analysis. In *International Semantic Web Conference (ISWC)*, pages 278–293. Springer.

Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. 2017. Knowledge Base Completion: Baselines Strike Back. In *Proceedings the Workshop on Representation Learning for NLP (Rep4NLP@ACL)*, page 69–74.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. 2021. Dynabench: Rethinking Benchmarking in NLP. In

Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), pages 4110–4124.

Bhushan Kotnis, Kiril Gashteovski, Julia Gastinger, Giuseppe Serra, Francesco Alesiani, Timo Sztyler, Ammar Shaker, Na Gong, Carolin Lawrence, and Zhao Xu. 2022a. Human-Centric Research for NLP: Towards a Definition and Guiding Questions. *arXiv preprint arXiv:2207.04447*.

Bhushan Kotnis, Kiril Gashteovski, Daniel Rubio, Ammar Shaker, Vanesa Rodriguez-Tembras, Makoto Takamoto, Mathias Niepert, and Carolin Lawrence. 2022b. MILIE: Modular & Iterative Multilingual Open Information Extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6939–6950.

Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Hung Ngo Quoc, Tuan Tran Nhat, and Manfred Hauswirth. 2016. The Graph of Things: A Step Towards the Live Knowledge Graph of Connected Things. *Journal of Web Semantics*, 37:25–35.

Pengfei Liu, Jinlan Fu, Yang Xiao, Weizhe Yuan, Shuaicheng Chang, Junqi Dai, Yixin Liu, Zihuiwen Ye, and Graham Neubig. 2021. ExplainaBoard: An Explainable Leaderboard for NLP. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL): System Demonstrations*, pages 280—-289.

Robert IV L. 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 *Proceesings of the Annual Meeting on Association for Computational Linguistics (ACL)*, page 5962–5971.

Christopher Malon, Kai Li, and Erik Kruus. 2022. Fast few-shot debugging for NLU test suites. In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 79–86. Association for Computational Linguistics.

Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. 2018. Fine-grained Evaluation of Rule-and Embedding-Based Systems for Knowledge Graph Completion. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 3–20.

George A. Miller. 1992. WordNet: A Lexical Database for English. *Communications of the ACM*, 38:39–41.

Pasquale Minervini, Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito. 2015. Efficient learning of entity and predicate embeddings for link prediction in knowledge graphs. In *Proceedings of the International Workshop on Uncertainty Reasoning for the Semantic Web (URSW@ISWC)*, volume 1479 of *CEUR Workshop Proceedings*, pages 26–37.

Aisha Mohamed, Shameem Parambath, Zoi Kaoudi, and Ashraf Aboulnaga. 2020. Popularity Agnostic Evaluation of Knowledge Graph Embeddings. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 1059–1068.

Shinichi Nakagawa and Innes C Cuthill. 2007. Effect size, confidence interval and statistical significance: A practical guide for biologists. *Biological reviews*, 82(4):591–605.

Avanika Narayan, Piero Molino, Karan Goel, Willie Neiswanger, and Christopher Re. 2021. Personalized Benchmarking with the Ludwig Benchmarking Toolkit. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

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

Matteo Palmonari and Pasquale Minervini. 2020. Knowledge Graph Embeddings and Explainable AI. *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges*, 47:49.

Yanhui Peng and Jing Zhang. 2020. LineaRE: Simple but Powerful Knowledge Graph Embedding for Link Prediction. In *IEEE International Conference on Data Mining (ICDM)*, pages 422–431. IEEE.

Priya Radhakrishnan, Partha Talukdar, and Vasudeva Varma. 2018. ELDEN: Improved Entity Linking Using Densified Knowledge Graphs. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1844–1853.

Thomas Rebele, Fabian Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. 2016. YAGO: A Multilingual Knowledge Base from Wikipedia, WordNet, and GeoNames. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 177–185. Springer.

Marco Tulio Ribeiro, Tongshuang Sherry Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, page 4902–4912.

Wiem Ben Rim, Carolin Lawrence, Kiril Gashteovski, Mathias Niepert, and Naoaki Okazaki. 2021. Behavioral Testing of Knowledge Graph Embedding Models for Link Prediction. In *Proceedings of the Conference on Automated Knowledge Base Construction (AKBC)*.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2020. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings.

In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Tara Safavi and Danai Koutra. 2020. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 8328–8350.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-Sequence Knowledge Graph Completion and Question Answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, page 2814–2828.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

Tristan Thrush, K. N. Bharadwaj Tirumala, Anmol Gupta, Max Bartolo, Pedro Rodriguez, Tariq Kane, William Gaviria Rojas, Peter Mattson, Adina Williams, and Douwe Kiela. 2022. Dynatask: A Framework for Creating Dynamic AI Benchmark Tasks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, pages 174–181.

Kristina Toutanova and Danqi Chen. 2015. Observed versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality@ACL*, pages 57–66.

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

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. 2019. On Evaluating Embedding Models for Knowledge Base Completion. In *Proceedings of the Workshop on Representation Learning for NLP (RepL4NLP@ACL)*, page 104–112.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1112–1119.

Gerhard Weikum, Xin Luna Dong, Simon Razniewski, Fabian Suchanek, et al. 2021. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. *Foundations and Trends® in Databases*, 10(2-4):108–490.

Yang Xiao, Jinlan Fu, Weizhe Yuan, Vijay Viswanathan, Zhoumianze Liu, Yixin Liu, Graham Neubig, and Pengfei Liu. 2022. DataLab: A Platform for Data Analysis and Intervention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, page 182–195.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. 2020. Learning Hierarchy-aware Knowledge Graph Embeddings for Link Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3065–3072.

Appendix

A Discussion on Related Work

A.1 Evaluation of NLP Models

Recent work in NLP has pointed to the problems of using single-score metrics (Ethayarajh and Jurafsky, 2020; Narayan et al., 2021; Bowman and Dahl, 2021; Kotnis et al., 2022a). In particular, such single-score metric evaluations do not expose the data-centric properties that were not learned by the models (e.g., problems with learning named entities that span across multiple tokens). In NLP, such issues are tackled by proposing multifaceted and/or explainable leaderboards and benchmarks (Liu et al., 2021; Gashteovski et al., 2022; Friedrich et al., 2022; Kotnis et al., 2022b; Xiao et al., 2022; Kiela et al., 2021; Thrush et al., 2022). To the best of our knowledge, there is no prior work that proposes such multi-faceted leaderboards for KG completion models.

A.2 Evaluation of KGC Models

KGC Benchmarks. Prior work has shown that learning latent representations—i.e., KG Embeddings (KGEs)—for the entities and relations is highly beneficial for tackling KGC tasks (Minervini et al., 2015; Ruffinelli et al., 2020). Researchers have evaluates the effects of hyperparameters (and different training strategies) of KGE models for link prediction. In particular, Kadlec et al. (2017) studied the effects of different training objectives on the DistMult model (Yang et al., 2015) and found that using the cross entropy loss function is a better alternative to the binary cross entropy. Ruffinelli et al. (2020) performed a systematic study on many KGE models with the LibKGE framework (Broscheit et al., 2020) and found that training components make a huge difference in model performance. Safavi and Koutra (2020) proposed a more data-centric benchmark (CoDEx), which improves upon previous benchmarks by proposing additional datasets extracted from Wikidata (Vrandečić and Krötzsch, 2014) and then using them to improve current KGE models. Sun et al. (2020) proposed an evaluation framework such that it breaks ties of same-score answers according to different strategies. PyKEEN (Ali et al., 2021) is another KGE framework and benchmark, which facilitates the training and evaluation of KGE models for KG completion tasks across a variety of datasets. Although highly useful for overall evaluation, none of these frameworks provide fine-grained evaluation of KGE models, nor an interactive interface for such evaluation.

Metrics. Other line of work focused on proposing different metrics for exposing different problems within the KGE models (Wang et al., 2019), though these metrics are averaged scores over the entire test set and do not provide additional insights about where the models make mistakes. Prior work has shown that the standard averaged evaluation metrics hits@k and MRR favor popular entities and relations (Mohamed et al., 2020). Contrary to such approaches, KGxBoard operates across the standard metrics (hits@k, MRR and MR), but in addition it supports highly customized fine-grained analysis and interactive interface.

Studying Specific Properties. Other line of work targets specific properties of the models, i.e., evaluating if a set of KGE models learned some specific properties. For example, relation symmetry is a property that has been extensively studied in the literature (Trouillon et al., 2016; Sun et al., 2019; Peng and Zhang, 2020; Zhang et al., 2020; Wang et al., 2014). Other line of work investigated other properties, such as entity type hierarchy (Rim et al., 2021; Zhang et al., 2020), (inverse) equivalence (Meilicke et al., 2018), subsumption (Meilicke et al., 2018), relation and entity frequency (Mohamed et al., 2020) and entity distribution (Bordes et al., 2013).

KG Analyzers. Recently, there were proposals for systems that analyse already constructed KGs (Ilievski et al., 2020). Such work, however, does not cover the evaluation of KGC models, it only covers analysis of already existing KGs.

B Details on Built-in Features

KGxBoard comes with built-in features that bucketize the data automatically. Some of the built-in features in KGxBoard are dataset-specific, while others are dataset-agnostic.

- Length of head/tail entity: the number of tokens in the head/tail entity (dataset-agnostic).
- Frequency of head/tail entity: the frequency of tail entity in the training set (dataset-agnostic).
- Frequency of the predicate: the frequency of the predicate in the training set (dataset-agnostic).
- Symmetry of relation: the symmetry of entity relations (dataset-specific; for now KGxBoard

supports FB15K-237).

- Entity type level: the most specific (highest) entity type level of true tail entity (dataset-specific, for now we support FB15K-237). In particular, we mapped each Freebase entity to its DBpedia (Auer et al., 2007) counterpart, and then used DBpedia's type information (Gangemi et al., 2012) in order to determine the most specific type level that is available in the data; e.g., if we have information *(Barack Obama; type; Person)* and *(Barack Obama; type; Politician)*, we use the latter because it is more specific.

C Details on Customized Buckets

To provide more fine-grained analysis that goes beyond the built-in features for bucketization, as well as to showcase the ability of KGxBoard to handle customized buckets, we partition the test data into buckets, based on the following customized features:

Buckets by relations. Each data point is placed in a bucket according to its relation. For example, the triples *(England; /location/location/contains; Lancaster)* and *(Los Angeles; /location/location/-contains; Beverly Hills)* are placed in the same bucket because they have the same relation. This bucketization is dataset-agnostic and we used it for both FB15K-237 and WN18RR.

Buckets by relation types 1-1, 1-M, M-1 or M-M. Following Bordes et al. (2013), we partition the data into four possible buckets according to their relation properties: 1-to-1 (1-1), 1-to-many (1-M), many-to-1 (M-1) and many-to-many (M-M). According to this definition, each relation has a property on how many entities it can have as a head or tail. For example, the relation *isAuthorOfBook* is 1-M relation (because one author can be the author of several books) and the relation *sportsTeamLocation* is 1-1 relation (because one sports team can have only one home location).

Tail entity type (level 1). The entity type (level 1, as described by Rim et al. (2021)) of the gold entity that needs to be predicted. This customized feature is specific for FB15K-237 and not for WN18RR. In particular, we leverage similar approach as with the built-in entity type level feature (described in Appendix B): we mapped each Freebase entity to its DBpeda counter part and then used DBpedia's type information to determine the type at level 1 of the entity type hierarchy.

Tail entity type (level 2). The entity type (level 2, as described by Rim et al. (2021)) of the gold entity that needs to be predicted. As with the previous customized feature, this customized feature is specific for FB15K-237 and not for WN18RR.

Relation's symmetry (for WN18RR). In Appendix B, we described KGxBoard's built-in feature that bucketizes the data into symmetric and asymmetric relations for the FB15K-237 dataset. This feature, however, is not natively supported by KGxBoard for the WN18RR dataset. For WN18RR, we followed Rim et al. (2021) and (1) got a set of all unique relations from WN18RR; (2) manually defined which relation is symmetric and which one is not. Then, we assigned a bucket (symmetric or asymmetric) to each data point in the test set.

D Training Details for PyKEEN and LibKGE

PyKEEN provides hyperparameters to reproduce the results of the original work of a given model, which we used to train RotatE, ConvE, TuckER, TransE and Rescal on both WN18RR and FB15K237 datasets. After training TransE, the hyperparameters for WN18RR returned a very low MRR result indicating that it failed to train for the task, and for DistMult, no configuration files corresponding to the above-mentioned datasets were provided. In such cases, we used insights from LibKGE (Ruffinelli et al., 2020) to train the models with PyKEEN to the best of our abilities; to implement the same models in the two frameworks, we match some parameters such as the number of epochs, embedding dimensions, initializer function, optimizer and learning rate arguments, including the scheduler and its parameters. However, the hyperparameters of a KGE model on the mentioned frameworks are not matched one-to-one. LibKGE allows the user to set both the initializer of the relation and entity embeddings along with setting the lookup embedder weight, patience, regularizer, dropout. In contrast, PyKEEN only allows the naming of the initializer function. Moreover, while negative sampling is possible in both frameworks, PyKEEN allows setting the negative sampler function that describes how to generate corrupt triples for training and allows the setting of negatives per positive triples as well as the filtering and corruption scheme, where LibKGE allows the previous as well as adding the number of samples for each head, relation, object. We expect that these differences in implementation and hyperparameters will have an impact on the results seen on KGcBoard. We provide the used hyperparameters and trained models with our code.

It is worth noting that PyKEEN does provide configurations for the same models with optimized parameters, but since the MRR and Hits@k results of these models are outperformed by the LibKGE models, we chose to make this comparison to both feature the difference in hyperparameters as well as the framework implementation in our experiments. For the models trained on LibKGE, we used the pretrained models obtained by Ruffinelli et al. (2020) as a result of hyperparameter optimization using LibKGE (Broscheit et al., 2020).

E Detailed Results for Bucketized Comparison of Models

In these experiments, we showcase how the users can use KGxBoard in order to compare one model trained on multiple hyperparameter settings. As explained in Section 5.3, on the one hand, we trained each model with the hyperparameters that aim to reproduce the work of the original papers (dubbed ORIGHP-MODEL); on the other hand we used pretrained models that use a hyperparameter optimization pipeline that ensures improved overall results (dubbed OPTIMHP-MODEL). Each model trained with the ORIGHP hyperparameter settings was trained with PyKEEN, by using the hyperparameters combination that aims to reproduce the results from the original papers and was proposed in PyKEEN's documentation. Each model trained with the OPTIMHP hyperparameter settings was trained with LibKGE, by using the hyperparameter combination proposed by Ruffinelli et al. (2020). The results are summarized in Table 3.

Model	Overall rank (OPTIMHP / ORIGHP)	MRR score (OPTIMHP / ORIGHP)	$b^{=}$	$b^{\neq}$
		FB15K-237		
ConvE	1 / 2	0.44 / 0.39	0.88	0.12
Rescal	1 / 2	0.45 / 0.38	0.83	0.17
DistMult	1 / 2	0.44 / 0.31	0.91	0.09
TransE	1 / 2	0.42 / 0.19	0.98	0.02
		WN18RR		
ConvE	1 / 2	0.46 / 0.28	1.00	0.00
Rescal	1 / 2	0.48 / 0.29	1.00	0.00
DistMult	1 / 2	0.48 / 0.27	1.00	0.00
TransE	1 / 2	0.24 / 0.13	0.91	0.09

Table 3: Comparison of KGC models trained with different hyperparameter settings. The shown results compare each model between its OPTIMHP and ORIGHP hyperparameter settings. The models trained with OPTIMHP are trained with hyperparameter optimization pipeline that ensures improved overall results. The models trained with ORIGHP use hyperparamter settings that aim to replicate the results from the original papers. 1 in *Overall rank* indicates better rank. The models were trained and tested on the FB15K-237 and WN18RR datasets. $b^{=}/b^{\neq}$ indicates the fraction of buckets where the overall rank is equivalent/not equivalent as the bucket's rank.

While the OPTIMHP models always outperform their ORIGHP counterparts in both FB15K-237 and WN18RR datasets, we still observed many cases where the ranking of the models flipped on some buckets. For example, OPTIMHP-CONVE on FB15K-237 performs better than ORIGHP-CONVE on the overall score, as well as on 88% of all the buckets. However, ORIGHP-CONVE performs better than OPTIMHP-CONVE for several buckets, including (1) the FB15K-237 triples with relation between award ceremony and award winner; (2) FB15K-237 triples whose tail entity types are of type "musical work". In a more extreme case for FB15K-237, when we compared OPTIMHP-TRANSE with ORIGHP-TRANSE, the optimized OPTIMHP-TRANSE is ranked as 1st on the overall score and in 98% of the buckets (this is to be expected, given that the difference in the overall score is 23 percentage points). Yet, in 2% of the buckets (4 buckets in total), ORIGHP-TRANSE is ranked as 1st.

F Detailed Results for Debugging

Table 4 shows debugging results for ConvE, Tucker, RotatE, and Rescal. In all cases, naive debugging improves Hits@1 on the held-out test debugging examples, and in-danger debugging often yields a further improvement. In cases where debugging Hits@5 and Hits@10 were high to begin with, debugging sometimes worsens these metrics, because most of the debugging examples will be teaching the model to swap the order of examples already in the top 5 or 10, rather than bring something new into the top 5 or 10 hits.

Only on ConvE does the naive method reduce original Hits@1 by more than 1%. This impact is possible because ConvE (and Tucker) have interaction parameters that can affect other relations, where RotatE and Rescal have only relation and entity embeddings, so that the debugging affects only

Model / relation	Hits@1	Hits@5	Hits@10	MR	MRR
ConvE/dated					
			Debugging test		
Before debugging	.0000	.9062	1.0000	3.3125	.3556
Naive	.0625	.3125	.5312	11.7188	.1863
In-danger	.1562	.7812	.9688	3.9688	.3943
			Original test		
Before debugging	.2710	.4734	.5690	186.0528	.3677
Naive	.2416	.4464	.5433	237.9980	.3402
In-danger	.2574	.4614	.5595	203.3516	.3546
Tucker/adjoins					
			Debugging test		
Before debugging	.0000	.8542	.9583	3.7708	.3895
Naive	.1875	.8542	.9167	5.0208	.4749
In-danger	.2083	.8333	.9167	6.7917	.4678
			Original test		
Before debugging	.3108	.5396	.6283	106.4038	.4171
Naive	.3010	.5262	.6162	126.6328	.4055
In-danger	.3004	.5232	.6140	119.0497	.4041
RotatE/friendship					
			Debugging test		
Before debugging	.0000	.0000	.0000	41.0465	.0419
Naive	.0465	.1860	.4302	18.9651	.1493
In-danger	.0465	.1860	.4535	21.2791	.1468
			Original test		
Before debugging	.2627	.4605	.5432	346.0619	.3554
Naive	.2627	.4605	.5432	345.7814	.3554
In-danger	.2627	.4605	.5432	345.8078	.3554
Rescal/adjoins					
			Debugging test		
Before debugging	.0000	.7761	.9030	5.5597	.3385
Naive	.1642	.7313	.8507	9.2687	.4038
In-danger	.2015	.6716	.8060	9.1194	.4040
			Original test		
Before debugging	.2596	.4665	.5562	295.5532	.3574
Naive	.2594	.4665	.5560	295.6199	.3573
In-danger	.2594	.4665	.5560	295.7509	.3573

Table 4: Debugging results for the relations with most symmetry violations.

the relation being debugged. On ConvE, the in-danger method reduces original Hits@1, 5, and 10 by about half as much as the naive method, while significantly improving all metrics on the debugging set. For the other models, the naive method can achieve simple and effective debugging.