# Hardware-agnostic Computation for Large-scale Knowledge Graph Embeddings

**Caglar Demir**[1] **and Axel-Cyrille Ngonga Ngomo**
Data Science Group, Paderborn University, Germany

## Abstract

Knowledge graph embedding research has mainly focused on learning continuous representations of knowledge graphs towards the link prediction problem. Recently developed frameworks can be effectively applied in research related applications. Yet, these frameworks do not fulfill many requirements of real-world applications. As the size of the knowledge graph grows, moving computation from a commodity computer to a cluster of computers in these frameworks becomes more challenging. Finding suitable hyperparameter settings w.r.t. time and computational budgets are left to practitioners. In addition, the continual learning aspect in knowledge graph embedding frameworks is often ignored, although continual learning plays an important role in many real-world (deep) learning-driven applications. Arguably, these limitations explain the lack of publicly available knowledge graph embedding models for large knowledge graphs. We developed a framework based on the frameworks DASK, Pytorch Lightning and Hugging Face to compute embeddings for large-scale knowledge graphs in a hardware-agnostic manner, which is able to address real-world challenges pertaining to the scale of real application. We provide an open-source version of our framework along with a hub of pre-trained models having more than 11.4 B parameters[2].

**Keywords** Knowledge Graph Embeddings, Hardware-agnostic Computation, Continual Training

**Code metadata**

| Nr. | Code metadata description | Please fill in this column |
|---|---|---|
| C1 | Current code version | v3 |
| C2 | Permanent link to code/repository used for this code version | https://github.com/dice-group/dice-embeddings |
| C3 | Permanent link to Reproducible Capsule | https://codeocean.com/capsule/6862303/tree/v1 |
| C4 | Legal Code License | AGPL-3.0 license |
| C5 | Code versioning system used | git, gitflow |
| C6 | Software code languages, tools, and services used | Python, Pytorch, Pytorch-Lightning, DASK, Hugging Face, Pandas, Numpy, and more |
| C7 | Compilation requirements, operating environments & dependencies | https://github.com/dice-group/dice-embeddings#installation |
| C8 | If available Link to developer documentation/manual | https://github.com/dice-group/dice-embeddings#documentation |
| C9 | Support email for questions | caglar.demir@upb and caglardemir8@gmail.com |

## 1. Introduction

Knowledge Graphs (KGs) represent structured collections of facts [1] and are being used in many challenging applications, including web search, question answering, and recommender systems [2]. Despite their usefulness in many applications, most knowledge graphs are incomplete, i.e., contain missing links. The task of identifying missing links in knowledge graphs is referred to as *link prediction*. In the last decade, a plethora of Knowledge Graph Embedding (KGE) approaches have been successfully applied to tackle various tasks including the link prediction task [2]. KGE models aim to learn continuous vector representations (embeddings) for entities and relations tailored towards the link prediction task.

[1] Corresponding Author

[2] https://github.com/dice-group/dice-embeddings

**Knowledge Graph Embeddings and Challenges:** Let $\mathcal{E}$ and $\mathcal{R}$ represent the sets of entities and relations. A KG is often formalised as a set of triples $\mathcal{G} = \{(\mathtt{h}, \mathtt{r}, \mathtt{t})\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ where each triple contains two entities $\mathtt{h}, \mathtt{t} \in \mathcal{E}$ and a relation $\mathtt{r} \in \mathcal{R}$ [3, 2]. Most KGE models are defined as a parametrized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ such that $\phi_\Theta(\mathtt{h}, \mathtt{r}, \mathtt{t})$ ideally signals the likelihood of $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ is true [2]. In a simple setting, $\Theta$ contains an entity embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$ and a relation embedding matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$, where $d$ stands for the embedding vector size. Given the triples ($\mathtt{Barack}$, $\mathtt{Married}$, $\mathtt{Michelle}$) and $(\mathtt{Michelle}, \mathtt{HasChild}, \mathtt{Malia}) \in \mathcal{G}$, a good scoring function is expected to return high scores for ($\mathtt{Barack}$, $\mathtt{HasChild}$, $\mathtt{Malia}$) and ($\mathtt{Michelle}$, $\mathtt{Married}$, $\mathtt{Barack}$), while returning a considerably lower score for ($\mathtt{Malia}$, $\mathtt{HasChild}$, $\mathtt{Barack}$). To compute a single score, embeddings of entities and relations are retrieved from $\mathbf{E}$, $\mathbf{R}$ and trilinear d-dimension vector multiplication is performed, i.e., $\phi(\mathtt{Barack}, \mathtt{HasChild}, \mathtt{Malia}) = \mathbf{Barack} \circ \mathbf{HasChild} \cdot \mathbf{Malia}$ (see [4]).

As $|\mathcal{G}|$ increases, the total training time of learning good representations $\Theta$ increases. This magnifies the importance of effective parallelism. This is often realised as data parallelism which stores a copy of a KGE model in available CPUs or GPUs. Most available frameworks including KGE frameworks rely on this paradigm (see PyKEEN [5] and libkge [6]. A $|\mathcal{G}|$, $|\mathcal{E}|$ and $|\mathcal{R}|$ grow, $\Theta$ often does not fit in a GPU. This limitation gives a rise to model parallelism and sharded training. Through FairScale[7] or DeepSpeed techniques [8] provided within Pytorch Lightning [9], our framework effectively partitions $\Theta$ into available CPUs and GPUs, instead of plain cloning. This allows to train gigantic models (>11 B parameters). PyKEEN [5] and libkge [6] lack of the model parallelism feature among many other features such as the deployment service.

## 2. Description

**Hardware-agnostic Computation:** The core goal of our framework is to facilitate learning large-scale knowledge graph embeddings in an hardware-agnostic manner. Hence, practitioners can use our framework to learn embeddings of KG on commodity hardware as well as a cluster of computers without chaining a single line of code. We based our framework on Pytorch Lightning [9] and DASK [10]. Pytorch Lightning allows our framework to use multi-CPUs,-GPU and even -TPUs in an hardware-agnostic manner. This implies that many important decisions at finding a suitable configuration for the learning process can be made automatically, i.e., finding a batch size that optimally fits in to the memory, scaling to cluster of computers. We observe that most embedding frameworks rely on a single core while reading and preprocessing the input KG. As the size of KG grows, this design decision becomes an increasing hindrance to scalability and increases the total runtime. Moreover, the process of the reading, preprocessing, and indexing an input KG is often intransparent to practitioners. That means that as the size of KG grows, practitioners are not informed about the current stage of the total computation. Through a DASK dashboard, our framework shares details about all steps of computation with users. The DASK dashboard can be used to used to analyse the reading and preprocessing steps in a fine-grained manner when computation is moved from a commodity computer to cluster of computers.
**Finding suitable configuration:** Our framework dynamically suggests a suitable configuration setting for a given dataset and available computational resources. This includes many features, e.g., finding most memory efficient integer data type for indexing, batch size, embedding vector size as well as learning rate for a given input configuration. Currently, we are working on forging our framework with Auto-Machine Learning techniques to facilitate the usage of framework by novice users. By this, we aim to share our expert knowledge with practitioners to that their computational and time budgets can be effectively utilized. Computational and time budgets of practitioners play an important role in real-world successful Machine Learning (ML) applications [11].

**Continual Learning and Deployment:** Our framework continues to assist practitioners after the embedding learning process. In many ML applications, the input data evolves with the time. Hence, continual learning plays an important role in successful applications of ML models[12]. Yet, most KGE frameworks do not provide means for continual learning. To alleviate this issue, our framework can be used to train KGE models on non-static data. Moreover, practitioners can deploy their model in a web-application without writing a single line of code as in our github repostiory.

**Extendability:** The software design of our framework allows practitioners to solely focus on their novel ideas, instead of engineering. For instance, a new model can be implemented in our framework without an effort. Inheriting from BaseKGE class, a new embedding model can be readily included into our framework.

```python
class ComplEx(BaseKGE):
    def __init__(self, args):
        super().__init__(args)
        self.name = 'ComplEx'
    def forward_triples(self, x: torch.Tensor) -> torch.Tensor:
        # (1) Retrieve Embedding Vectors
        head_ent_emb, rel_ent_emb, tail_ent_emb = self.get_triple_representation(x)
        # (2) Split (1) into real and imaginary parts.
        emb_head_real, emb_head_imag = torch.hsplit(head_ent_emb, 2)
        emb_rel_real, emb_rel_imag = torch.hsplit(rel_ent_emb, 2)
        emb_tail_real, emb_tail_imag = torch.hsplit(tail_ent_emb, 2)
        # (3) Compute Hermitian inner product.
        real_real_real = (emb_head_real * emb_rel_real * emb_tail_real).sum(dim=1)
        real_imag_imag = (emb_head_real * emb_rel_imag * emb_tail_imag).sum(dim=1)
        imag_real_imag = (emb_head_imag * emb_rel_real * emb_tail_imag).sum(dim=1)
        imag_imag_real = (emb_head_imag * emb_rel_imag * emb_tail_real).sum(dim=1)
        return real_real_real + real_imag_imag + imag_real_imag - imag_imag_real
```

Figure 1: Including an implementation of state-of-the-art Embedding model in our framework.

**Summary of Initial Experimental Results:** We used the most recent DBpedia 2021 benchmark dataset[3] to evaluate our framework in depth. Our experiments suggest that a state-of-the-art KGE model with more than 11.4B parameters can be successfully trained and applied in link prediction, and relation prediction [4]. We refer to the project page for the details and log files about pretrained models.

## 3. Software Impact

Our framework facilitates the use of KGE models on large KGs without requiring expert knowledge in software engineering. Hence, it helps practitioners to spend more time on generating value by using embedding models, instead of investing it into engineering for large-scale experiments.Our framework is now being deployed in real use cases within the funded research projects mentioned in the acknowledgements.

**Limitations and future improvements:** We plan to investigate (1) pseudo-labeling to leverage unlabelled data, (2) Auto-ML for the embedding model design, and (3) state-of-the-art continue learning techniques.

**Scholarly Publications:** Our framework have been effectively used to learn knowledge graphs embeddings in several published works [13],[3],[14],[4],[15],[16], and [17].

**Acknowledgements:** This work has been supported by the German Federal Ministry of Education and Research (BMBF) within the project DAIKIRI under the grant no 01IS19085B and by the the German Federal Ministry for Economic Affairs and Energy (BMWi) within the project RAKI under the grant no 01MD19012B.

## References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, et al., Knowledge graphs, arXiv preprint arXiv:2003.02320 (2020).

[2] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (1) (2015) 11–33.

[3] C. Demir, A.-C. N. Ngomo, Convolutional complex knowledge graph embeddings, in: Eighteenth Extended Semantic Web Conference - Research Track, 2021.

[4] C. Demir, J. Lienen, A.-C. Ngonga Ngomo, Kronecker decomposition for knowledge graph embeddings, in: Proceedings of the 33rd ACM Conference on Hypertext and Social Media, HT '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 1–10.

[3] https://databus.dbpedia.org/dbpedia/collections/dbpedia-snapshot-2021-06

[4] https://hobbitdata.informatik.uni-leipzig.de/KGE/DBpediaQMultEmbeddings_03_07/

[5] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, J. Lehmann, PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings, Journal of Machine Learning Research 22 (82) (2021) 1–6.

[6] S. Broscheit, D. Ruffinelli, A. Kochsiek, P. Betz, R. Gemulla, LibKGE - A knowledge graph embedding library for reproducible research, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, pp. 165–174.

[7] M. Baines, S. Bhosale, V. Caggiano, N. Goyal, S. Goyal, M. Ott, B. Lefaudeux, V. Liptchinsky, M. Rabbat, S. Sheiffer, A. Sridhar, M. Xu, Fairscale: A general purpose modular pytorch library for high performance and large scale training (2021).

[8] J. Rasley, S. Rajbhandari, O. Ruwase, Y. He, Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 3505–3506.

[9] W. Falcon, et al., Pytorch lightning, GitHub. Note: https://github. com/PyTorchLightning/pytorch-lightning 3 (2019) 6.

[10] M. Rocklin, Dask: Parallel computation with blocked algorithms and task scheduling, in: K. Huff, J. Bergstra (Eds.), Proceedings of the 14th Python in Science Conference, 2015, pp. 130 – 136.

[11] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, Advances in neural information processing systems 20 (2007).

[12] T. Diethe, T. Borchert, E. Thereska, B. Balle, N. Lawrence, Continual learning in practice, arXiv preprint arXiv:1903.05202 (2019).

[13] C. Demir, D. Moussallem, S. Heindorf, A.-C. Ngonga Ngomo, Convolutional hypercomplex embeddings for link prediction, in: V. N. Balasubramanian, I. Tsang (Eds.), Proceedings of The 13th Asian Conference on Machine Learning, Vol. 157 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 656–671.

[14] C. Demir, D. Moussallem, A.-C. N. Ngomo, A shallow neural model for relation prediction, in: 2021 IEEE 15th International Conference on Semantic Computing (ICSC), IEEE, 2021, pp. 179–182.

[15] H. M. Zahera, S. Heindorf, S. Balke, J. Haupt, M. Voigt, C. Walter, F. Witter, A.-C. Ngonga Ngomo, Tab2onto: Unsupervised semantification with knowledge graph embeddings, in: ESWC, 2022.

[16] N. Kouagou, S. Heindorf, C. Demir, A.-C. N. Ngomo, Learning concept lengths accelerates concept learning in alc, in: Nineteenth Extended Semantic Web Conference - Research Track, Springer, 2022.

[17] S. Heindorf, L. Blubaum, N. Dusterhus, T. Werner, V. Golani Nandkumar, C. Demir, A.-C. Ngonga Ngomo, Evolearner: Learning description logics with evolutionary algorithms, in: WWW, ACM, 2022, pp. 818–828.