2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI) | 978-1-6654-4135-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/RTSI50628.2021.9597228

Exploring the Runtime Performance of Knowledge Graph Embedding Methods

Angelica S. Valeriani, Guido Walter Di Donato, Marco D. Santambrogio
Dipartimento di Elettronica, Informatica e Bioingegneria
Politecnico di Milano, Milan, Italy
angelica.valeriani@mail.polimi.it
{guidowalter.didonato, marco.santambrogio}@polimi.it

***Abstract*—In recent years, Knowledge Graphs (KGs) have become ubiquitous, powering recommendation systems, natural language processing, and query answering, among others. Moreover, representation learning on graphs has reached unprecedentedly effective graph mining. In particular, Knowledge Graph Embedding (KGE) methods have gained increasing attention due to their effectiveness in representing real-world structured information while preserving relevant properties. Current research mainly focuses on improving and comparing the effectiveness of new KGE models on different predictive tasks. However, the application of KGE techniques in the industrial scenario sets a series of requirements on the runtime performance of the employed models. For this reason, this work aims to enable an effortless characterization of the runtime performance of KGE methods in terms of memory footprint and execution time. To this extent, we propose *KGE-Perf*, a framework for evaluating available state-of-the-art implementations of KGE models against graphs with different properties, focusing on the efficacy of the adopted optimization strategies. Experimental evaluation of three representative KGE algorithms on open-access KGs shows that multi-threading on CPU is effective, but its benefits decrease as the number of threads grows. The usage of vectorized instruction shows encouraging results in speeding up the training of KGE models, but GPU proves, hands down, to be the best architecture for the given task. Moreover, experimental results show how the RAM usage strongly depends on the input KG, with only slight variations between different models or hardware configurations.**

***Index Terms*—Knowledge Graphs, Knowledge Graph Embedding, Performance, Multi-Threading, Vectorization, GPU**

I. Introduction

As information is becoming more and more structured, Knowledge Graphs (KGs) are gathering increasing interest as they enable structured representations of real world information. Their widespread application in industry and academia brought to a remarkable research effort towards large-scale information extraction and manipulation from multiple sources. KGs can be used to model complex biological systems [1], also for medical purpose, e.g. considering COVID-19 biology and pathophysiology [2]. They also found application in e-commerce (to capture items, relations - as advertisement, search ranking, marketing - and user problems [3]), in recommender systems is (as they can merge multiple data sources to integrate external knowledge [4]) and in many Natural Language Processing (NLP) applications (e.g. analysis of legal documents in the field of journalism and law [5]). Although KGs are effective in representing structured data, their manipulation is hard to be managed at a practical level. This is where Knowledge Graph Embedding (KGE) comes into play, allowing to simplify data manipulation while preserving the KG structure. KGE methods aim at embedding KG components in a vector space, reducing memory utilization and computational complexity as vector operations are simpler and faster than comparable operations on graphs. Moreover vector spaces are more amenable to data science than graphs, thus KGE is particularly promising for emerging Machine Learning (ML) applications, like KG completion [6], or NLP [5].

Current research on KGE mainly focuses on models' accuracy, a fundamental metric for ML tasks. Unfortunately, this advancement does not go in parallel with the pursuit of performance, which is indeed crucial for the application of these techniques in real industrial scenarios. In particular, an analysis of the runtime performance of KGE methods in relation to the input graph's properties and the employed computational resources was still missing in the literature. For this reason, in this work we propose *KGE-Perf*, a framework to evaluate in a simple way models' performance (train/inference time and memory usage) on different hardware configurations (i.e., number of CPU threads, usage of vectorized instructions, usage of GPU) to support industry in decision-making regarding the employed KGE models and computational facilities. The proposed framework is built on top of Ampligraph [7] to guarantee comparability and uniformity among different methods, and to minimize the bias introduced by implementation choices. Moreover, here we present the experimental results of an extensive analysis (more than 630 hours of total execution time) obtained through the proposed framework. To provide a significant overview, three of the most representative KGE models in the state-of-the-art have been employed in our experiments: *TransE* [8], *DistMult* [9] and *ConvKB* [10]. The models have been evaluated on five widely employed open-access KGs to guarantee replicability and comparability.

The remainder of the paper is organized as follows. Section II furtherly details the purpose of this work, describes the related work, and provides the background on KGE models, with a focus on the models employed in our experiments. Section III describes the proposed framework and its implementation, while in Section IV the experimental results are illustrated and discussed. Finally, Section V draws the conclusions and outlines the possible future work.

II. Background and Related Work

In the last few years, Knowledge Graphs such as WordNet [11], Freebase [12], and YAGO3 [13] have been playing a fundamental role in many Artificial Intelligence (AI) applications, like Relation Extraction (RE) or Question Answering (Q&A). They contain huge amounts of structured data in form of triples <*head, relation, tail*>, denoted as <*h,r,t*>, where the relation (i.e., a link) models the relationship between the two entities (i.e., two nodes). Due to their structure's versability, KGs are becoming ubiquitous, powering enhanced query processing, conversational applications, and recommender system, among the others [14]. In particular, Graph Machine Learning on KGs is gathering increasing interest, also due to the fact that representation learning on graphs has reached unprecedentedly effective graph mining capabilities [15]. Representation learning is done through Knowledge Graph Embedding models, able to learn a mapping from a network to a vector space, while preserving relevant properties, like the network topology or vertex-to-vertex relationships.

Despite the already astonishing performance of existing models, new KGE methods are being developed continuously, in order to further improve on the achieved effectiveness. In the vast pool of available embedding models, studies providing a comparison between different techniques are crucial to guide the selection of the most appropriate method for a certain predictive task. For example, a survey considering the tasks of graph completion and Link Prediction (LP) is proposed in [16], where metrics as Precision and Mean Average Precision (MAP) are considered. LP task is described and analyzed also in [17], where hits are considered to describe model's performance and a theoretical analysis about the relation between model's properties and obtained results is provided. A similar analysis is performed in [6], in which not only hits are evaluated, but also Mean Rank (MR) and Mean Reciprocal Rank (MRR), commonly employed metrics in LP. However, to the best of our knowledge, previous comparative studies mainly focused on the predictive power of the investigated models, without properly considering their runtime performance. Indeed, exploring this aspect is fundamental to unleash the great potential of KGs and KGE in the industrial scenario. For this reason, in this work we propose a framework to easily characterize and compare the performance of different embedding models, in relation to the properties of the input KG and the employed hardware configuration. Here we also describe the experimental results obtained through *KGE-Perf*, providing a comparison between the most representative methods in the three categories of KGE models, classified by the triple matching score method. In detail, in this work we have explored *TransE* from Geometric Models (GMs), *DistMult* from Tensor Decomposition Models (TDMs), and *ConvKB* from Deep Learning Models (DLMs).

The remainder of this Section provides an overview of the three categories of existing representation learning methods. It also present a description of the three KGE models explored in our experimental evaluation.

A. Geometric Models

Geometric Models aim to find vector representations of KG entities in an embedding space by regarding relations as geometric transformation of the entities in the latent space. In these models, the head entity embedding undergoes a spatial transformation depending on the values of the relation embedding. Then, the fact score is computed as the distance between the resulting vector (i.e., translated image of head) and the tail embedding vector [14]. GMs are classified in three groups: *Pure Translational*, *Transitional with Additional Embeddings*, and *Roto-translational*.

1) TransE: is the simplest and the most representative geometric KGE model [8], [14]. As every other pure translational model, it represents all vectors of entities and relations as one-dimensional vectors lying on a single vector space. The primary idea behind *TransE* is that the relation is interpreted as a translation vector **r**, so that **h + r** $\approx$ **t** if the fact <*h, r, t*> holds, while **h + r** should be far away from **t** if the triple is false (i.e., a negative triple, denoted as <*h',r,t'*>). To learn such embedding, *TransE* minimizes a loss function $\mathcal{L}$ (Eq. 1), with a given margin (γ), computed as sum of dissimilarity measure d (L1 or L2 norm) over the training set [8].

$$\mathcal{L} = \sum_{(h,r,t)\in \mathbf{S}} \sum_{(h',r,t')\in \mathbf{S}'} [\gamma + d(h+r,t) - d(h'+r,t')]_+ \tag{1}$$

Despite having difficulties in handling one-to-many, many-to-one, symmetric and transitive relations, *TransE* results competitive in terms of accuracy when compared to other KGE models, while obtaining outstanding performance in terms of time convergence [14], [15].

B. Tensor Decomposition Models

Tensor Decomposition Models compute embedding vectors solving a tensor decomposition problem on a partially observable 3D adjacency matrix representing the input KG. Such a tensor is decomposed into a multi-linear product of low-dimensional embedding vectors of entities and relations. The score of each fact is computed operating the multi-linear product on the specific embeddings involved in that fact. Then, embedding vectors are learned as usual by optimizing the scoring function for all the training facts. TDMs are classified as *Bilinear* and *Non-bilinear* models, on the basis of the employed scoring function.

1) DistMult: is the simplest TDM and a semantic matching model [9]. It forces all relation embeddings to be diagonal matrices $\mathbf{r} \in \mathbb{R}^{d\times d}$, resulting in an easier model to train with a reduced space of parameters to be learnt. However, this makes the score function commutative, treating all the relations as symmetric. Formally, the scoring function is defined as:

$$f_r(h,t) = \mathbf{h}^\top diag(\mathbf{r})\mathbf{t} = \sum_{i=0}^{d-1} [\mathbf{r}]_i \cdot [\mathbf{h}]_i \cdot [\mathbf{t}]_i$$

Thanks to its simplicity and its competitive LP performance (when carefully tuned), *DistMult* is the most employed TDM in literature [14], [18].

C. Deep Learning Models

Deep Learning embedding models use nonlinear Neural Networks (NN) as universal approximators to compute triple matching score. NNs learn parameters that they combine with input data to recognize significant patterns [14]. Deep NNs organize parameters into separate layers, usually interspersed with non-linear activation functions. Numerous kinds of layers have been developed in time, based on very different operations they perform on the input data. DLMs usually learn embeddings jointly with the parameters of the layers, resulting in more expressive models, but also potentially heavier and harder to train. They are classified on the basis of the employed NN architecture: *Convolutional*, *Capsule*, and *Recurrent*.

1) ConvKB: is a convolutional model, that represents entities and relations as same-sized one-dimensional embedding vectors [10]. Each triple *<h,r,t>* is represented as a three-column matrix, in which each column represents the embedding of the respective triple element. It employees a convolutional NN with multiple filters to capture global relationships and transitional characteristics, resulting in an output feature map that is let through a dense layer to compute the fact score. This score is then used to predict the triple validity, that is the probability that the input fact is valid. Formally, the score function f is defined as:

$$f(h,r,t) = concat(g([\mathbf{v}_h, \mathbf{v}_r, \mathbf{v}_t] * \Omega)) \cdot \mathbf{w}, \tag{2}$$

where Ω and $\mathbf{w}$ are shared parameters, independent of h, r, and t, $*$ denotes a convolution operator, and *concat* denotes a concatenation operator. Experiments show that *ConvKB* achieves better LP performance than previous state-of-the-art models on several benchmark datasets, but at the expense of a slow time convergence [6], [14], [15].

III. Methodology and Implementation

As anticipated in Section I, the purpose of this work is to characterize the runtime performance of different KGE methods in relation to the input graphs' properties and the adopted optimization strategies. To this aim, we propose *KGE-Perf* (Figure 1), a python framework to easily handle different KGE experiments, supporting multiple graphs, models, and configuration settings. Such framework has been built on top of AmpliGraph [7], an open source library by Accenture for supervised learning on KGs. AmpliGraph is, in turn, based on TensorFlow [19], a state-of-the-art open source platform for ML, widely employed both in industry and academia. We chose to take the implementations of all the supported KGE models from a single framework, in order to guarantee consistency and comparability among different methods, trying to minimize the bias introduced by the implementation choices.

KGE-Perf allows users to configure the execution of KGE experiments by setting a series of input parameters, and produces informative visualizations of the experimental results to enable an easy comparison among different models and hardware configurations. As shown in Figure 1, there are two class of parameters: fixed parameters are the same for the whole set of experiments, while variable parameters assume

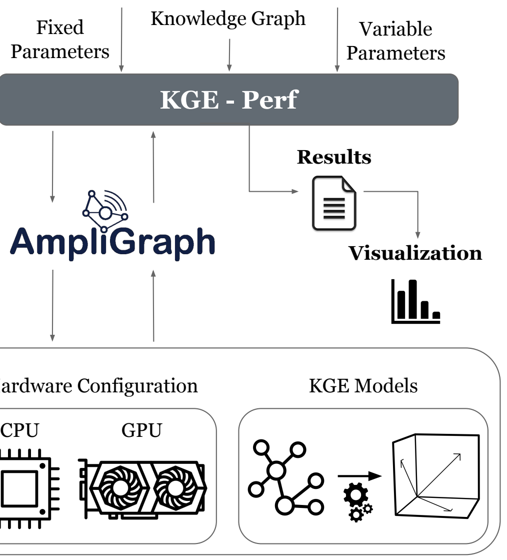

Fig. 1. Overview of the *KGE-Perf* framework.

different values and are tested in each possible configuration. Thus, the total number of executions in a set of experiments is given by the product of the number of different values each parameter can assume. The input graphs can be selected among the following open-access KGs, widely employed in the literature: *WN11, WN18, WN18RR, FB13, FB15k, FB15k-237, YAGO3-10, O*NET20K, PPI5K, NL27K,* and *CN15K*. Moreover, the user can eventually decide to employ custom KGs in the experiments, encoded as CSV/RDF files containing lists of triples in the form *<h,r,t>*. In this case, the user must also indicate the size (in terms of percentages of the total number of triples) of the train, validation, and test sets, respectively employed in the training of the KGE models (train and validation) and in their evaluation in inference phase (test).

For what concerns the KGE models to be employed in the experiments, the user can choose among all the different models available in AmpliGraph. The learning process can be configured by setting the following parameters:

- *Embedding space dimensionality:* number of components of the embedding vectors computed by the KGE model;
- *Number of epochs:* iterations of the training loop;
- *ETA:* number of negative samples that must be generated at runtime during training, for each positive sample;
- *Number of batches:* subsets in which the training set must be split during the training loop;
- *Loss function:* objective function to minimize during the training (options: pairwise/absolute max-margin, self adversarial sampling, (multiclass) negative log-likelihood, and binary cross entropy);
- *Optimizer:* method employed to minimize the loss function (options: stochastic gradient descent, Adam, Adagrad, Momentum)
- *Learning rate:* measure of the step size of the optimizer at each iteration;
- *Seed:* input for random numbers generator (replicability).

Finally, the user can select different hardware configurations to be tested in the experiments. *KGE-Perf* supports 3 different classes of configurations:

- *CPU:* employs a basic instance of the TensorFlow backend, leveraging only multi-threading for performance optimization;
- *VECT:* employs an instance of the TensorFlow backend that, in addition to multi-threading, leverages the vectorized instructions eventually available in the CPU;
- *GPU:* employs an instance of the TensorFlow backend that, in addition to multi-threading, leverages all the NVidia GPUs (using the CUDA programming model) eventually available in the system.

The different instances of the TensorFlow backend are handled through the Conda environment management system. For each class of HW configuration, the framework allows to set the number of CPU threads to employ in the experiments. In particular, the selected number of threads sets the TensorFlow's `intra_op_parallelism_threads` parameter, which defines the number of threads used to parallelize the execution of individual scheduled operations [19]. On the other hand, the `inter_op_parallelism_threads` parameter defines the size of the pool of threads different operations are enqueued on. *KGE-Perf* automatically sets the `inter_op_parallelism_threads` equal to the number of CPU sockets in the system, according to Intel's recommendations on how to maximize TensorFlow performance [20].

For each experiment, *KGE-Perf* evaluates the model's training time, as well as the inference time for entities, relations, and scores of the triples composing the test set. For each task, the framework measures both the wall-clock time (i.e., actual amount of time taken to perform a task) and the total CPU time (i.e., cumulative amount of time spent by each CPU thread to execute the program). The framework also monitors the memory consumption during the execution of the experiments. In particular, peak RAM usage is measured both immediately after loading the input graph and for the overall experiment. This allows to retrieve the values of memory utilization of the selected model for a given graph, ignoring the amount of memory employed for storing the graph itself. After the execution, the results of each experiment are saved as a row in a CSV file. Once all the experiments are completed, the results are employed to generate PDF files containing plots realized through the MatPlotLib and Seaborn libraries (see Figure 2 for a graphical example). To guarantee high easiness of usage, all the procedures have been automatized both for running the experiments and visualizing the obtained results. Nonetheless, the user can eventually utilize the computed results for custom visualization or further analysis.

IV. Experimental Evaluation

In this section, firstly we describe the experimental design and setup employed in the presented evaluation. Then, we present and discuss the experimental results obtained through the proposed framework.

TABLE I
Knowledge Graphs Properties

Graph	Entities	Relations	Train	Validation	Test
wn18	40943	18	141442	5000	5000
wn18rr	40943	11	86835	3034	3134
fb15k	14951	1345	483142	50000	59071
fb15k-237	14541	237	272115	17535	20466
yago3-10	123182	37	1079040	5000	5000

A. Experimental Design and Setup

As already introduced, in this work we have evaluated three of the most representative KGE models in the state-of-the-art: *TransE* from Geometric Models, *DistMult* from Tensor Decomposition Models, and *ConvKB* from Deep Learning Models. The models have been evaluated on five widely employed open-access KGs to guarantee replicability and comparability. We have chosen benchmark graphs with different features for our evaluation, in order to evaluate how such features impact on the runtime performance of the tested KGE models. *WN18* is a subset of WordNet [11], and it tends to follow a strictly hierarchical structure, as most of its triples consist of hyponym and hypernym relations. *FB15K* is a subset of Freebase [12], and a large fraction of content describes facts about movies, actors, awards and sport. *WN18RR* and *FB15K-237* are, respectively, subsets of *WN18* and *FB15K* deprived of inverse relations. Finally, *YAGO3-10* is a subset of the YAGO3 knowledge base [13], consisting of entities with a minimum of 10 relations each. Most of its triples deal with descriptive attributes of people, as citizenship, gender, and profession. Table I summarizes the properties of the selected KGs in terms of number of entities, relations, and triples in the train, validation, and test sets.

In our experiments, we tested each class of hardware configurations supported by the proposed framework, namely *CPU*, *VECT* and *GPU*. For each class of configurations, three experiments employing a different numbers of threads were run. Baseline experiments on CPU (without enabling vectorization) were conducted on a server equipped with two Intel Xeon E5-2680 v2 @2.8GHz (for a total of 20 cores and 40 threads) and 378GB of RAM. For such tests we employed a number of CPU threads equal to 8, 16 and 32. All the other tests (leveraging vectorized instructions or GPU) were run on a server equipped with an Intel Core i7-6700 @3.4GHz (4 cores, 8 threads), 32GB of RAM, and a Nvidia GeForce GTX 960 (1024 cores, 1126 MHz, 2 GB of GDDR5 @ 1750 MHz). For *VECT* and *GPU* tests we employed a number of CPU threads equal to 2, 4 and 8. The choice of running experiments leveraging vectorized instructions on the smaller server is tied to the fact that the Intel Core i7-6700 supports SSE4.1, SSE4.2, AVX, AVX2 and FMA instructions, while the Intel Xeon E5-2680 v2 only supports SSE4.1, SSE4.2 and AVX instructions, thus the impact of enabling vectorization on runtime performance would have been less evident. Table II lists the specific versions of the tools employed in our tests, for the sake of replicability and comparability.

TABLE II
EMPLOYED TOOLS WITH REPECTIVE VERSIONS

Tool	pandas	ampligraph	cuDNN	Keras-Applications
Version	1.0.3	1.3.1	7.6.5	1.0.8
Tool	numpy	tensorflow	CUDA Toolkit	Keras-Preprocessing
Version	1.18.1	1.15.0	10.0.130	1.1.0

TABLE III
LEARNING PROCESS SETTING

Fixed Parameter	**Value**
Embedding space dimensionality	256
Number of epochs	500
ETA	2
Number of batches	100
Loss function	pairwise, max-margin loss
Optimizer	Adam
Learning Rate	0.01

For each combination of KGE model, input graph, and hardware configuration, we repeated the experiment twice to reduce measurement bias. Thus, the total number of experiments (excluding preliminary tests) is 270, totalizing 630+ hours of wall-clock execution time. For what concerns the fixed parameters for the configuration of the learning process, after a background study and a number of preliminary tests, a common suitable setting for all KGE models was determined. The choice of selecting a common setting derives from the need of an efficient and consistent way to compare the performance of different models (trained on different graphs), across the investigated optimization strategies. For instance, from preliminary tests, we noticed how the choice of the optimizer greatly impact the runtime performance of the investigated models. The values of all the fixed parameters employed in the reported experiments are shown in Table III.

B. Experimental Results

As already introduced, we have employed *KGE-Perf* to monitor the training and inference times (in terms of both wall-clock and CPU time), as well as the memory utilization, in the considered set of KGE experiments. For reasons of space, in this paper we report a graphical representation of the obtained results only in terms of wall-clock training time (Figure 2). Nonetheless, here we comprehensively discuss the outcomes of our experiments, considering also the other measured runtime performance metrics. The CSV file containing all the computed results is available online [1], together with other plots automatically generated through the proposed framework.

Considering the scaling of the training time, Figure 2 shows how for both *CPU* and *VECT* configurations, the training time diminishes as the number of threads grows, for all the tested models. We can also notice how the benefits of increasing the number of CPU threads decrease as the number of threads grows. This is particularly true for the *CPU* configurations, where the speedups obtained when passing from 16 to 32 threads is negligible if compared to those obtained when passing from 8 to 16 threads. Instead, for the *VECT* configurations, passing from 4 to 8 threads reduces significantly the training time, even if the speedups are lower than those obtained when passing from 2 to 4 threads. These results shows how vectorization is much more efficient that multi-threading alone when optimizing the training of KGE models. This insight is confirmed by looking at the CPU training time, that significantly increases with the number of employed threads in *CPU* configurations, while resulting only in a slight reduction of the total elapsed time. Instead, in the *VECT* configurations, a trivial increase of the total CPU time is translated in a significant reduction of the total wall-clock time. For what concern *GPU* configurations, it is possible to see how both CPU and wall-clock time are not affected at all by the number of employed CPU threads, in accordance to TensorFlow's management of cores' internal division [20].

Comparing the different hardware configurations, we can see how the training time is always lower when using the GPU, compared to the tested *CPU* or *VECT* configurations. Moreover, for *DistMult* and *TransE* models, the *VECT* configuration with 2 threads is faster than the *CPU* configuration with 8 threads, for each input KG. This is not true for the *ConvKB* model, where the 8-threaded *CPU* configuration is faster than the 2-threaded *VECT* one. However, also for *ConvKB*, it is sufficient to pass to the 4-threaded *VECT* configuration to obtain much shorter training time than the 32-threaded *CPU* one, confirming the better scaling of *VECT* configurations w.r.t. the number of employed threads.

Comparing the different models, we can see how *DistMult* and *TransE* have similar performance on all graphs, with *TransE* being slightly faster on average. Instead, *ConvKB* is always significantly slower than the competitor models. Moreover, the performance of *ConvKB* seems to be much more affected by the input graph's properties than the other methods. Figure 2 shows how *DistMult* and *TransE* take almost the same amount of time to train on *WN18* and *WN18RR*, with an average $\sim 10\%$ reduction for the graph deprived of inverse relations, while *ConvKB* demonstrates an average speedup greater than 30%. Moreover, the reduction in the training time passing from *FB15K* to *FB15K-237* is of $\sim 30\%$ for both *DistMult* and *TransE*, while for *ConvKB* is greater than 40%. Finally, RAM utilization is constant across different *CPU* and *VECT* configurations, and it is variably ($\sim 10\%$ to $\sim 70\%$) lower for the *GPU* ones, for all the models and input graphs. Moreover, the memory usage of the different models is always comparable, and mainly depends on the size of the input graph in terms of both number of entities and training triples.

V. CONCLUSIONS AND FUTURE WORK

The runtime performance of KGE models is a crucial aspect to be investigated for fulfilling the potential of such methods in the industrial scenario. In this work we presented *KGE-Perf*, a user-friendly python framework that allows users to easily compare the runtime performance of various embedding models on KGs with different properties, running on specific hardware configurations. The proposed framework

[1]https://github.com/gwdidonato/KGE-Perf-Results

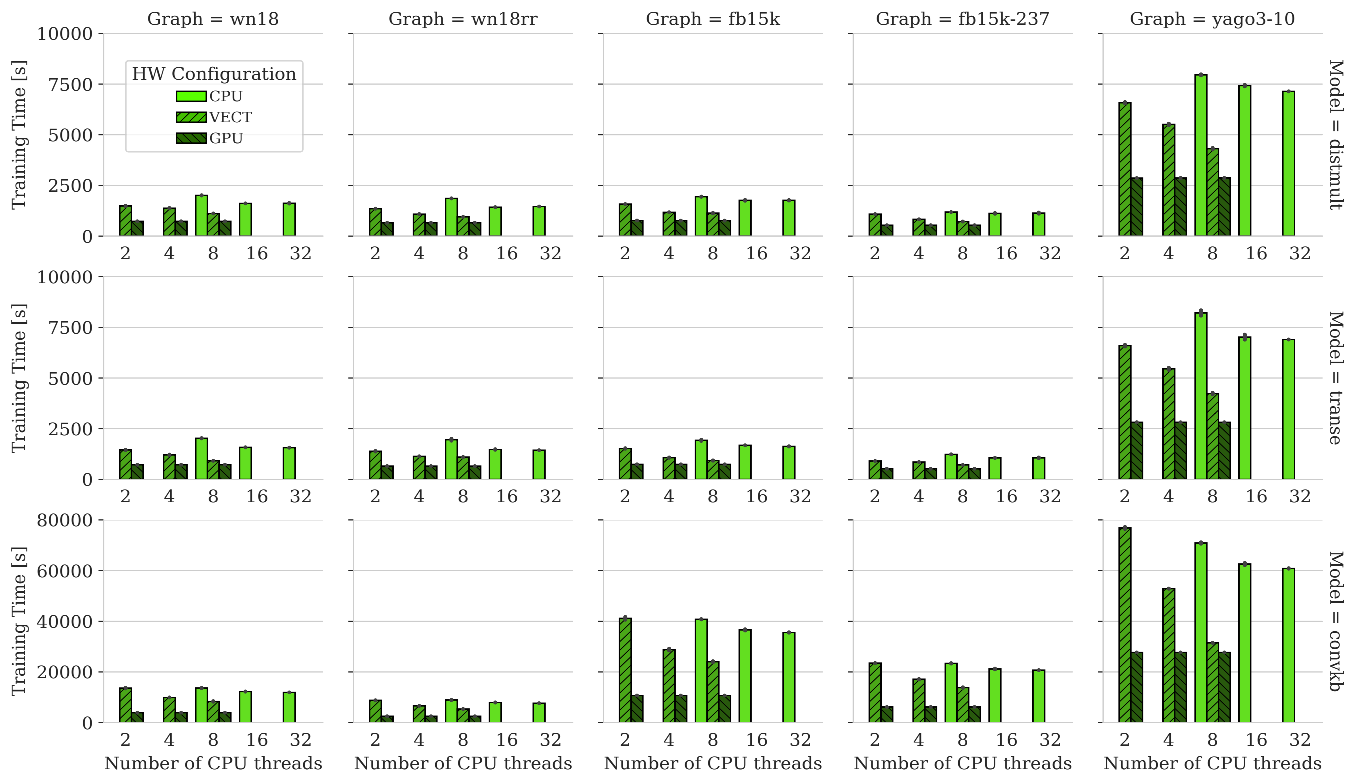

Fig. 2. Summary results in terms of total wall-clock time required for the training of the different KGE models.

accepts as input a series of fixed and variable parameters, and it automatically handles the execution of the given set of experiments and the visualization of the obtained results. We also presented the outcomes of an extensive experimental evaluation we conducted using *KGE-Perf*, testing 3 embedding models on 5 different KGs and a series of hardware configurations. Experimental results show that multi-threading on CPU is effective for reducing the training time of KGE models, but its benefits decrease as the number of threads grows. Vectorized instructions provide significant speedups when training embedding models, but GPU proves, hands down, to be the best architecture for the given task. The computed results also show that RAM consumption mainly depends on the input graph, with only slight variations among different models. Future work involves enabling to control the number of GPUs employed in *GPU* configurations, and integrating the assessment of the models' accuracy in the performance evaluation process. We also plan to openly release our implementation to the public in the near future.

References

[1] P. Ernst, C. Meng, A. Siu, and G. Weikum, "Knowlife: A knowledge graph for health and life sciences," in *2014 IEEE 30th International Conference on Data Engineering*, 2014, pp. 1254–1257.

[2] D. Domingo-Fernández *et al.*, "Covid-19 knowledge graph: a computable, multi-modal, cause-and-effect knowledge model of covid-19 pathophysiology," *bioRxiv*, 2020.

[3] F.-L. Li *et al.*, "Alimekg: Domain knowledge graph construction and application in e-commerce," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020.

[4] F. Zhang *et al.*, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

[5] A. Berven *et al.*, "A knowledge-graph platform for newsrooms," *Computers in Industry*, vol. 123, p. 103321, 2020.

[6] H. N. Tran and A. Takasu, "Analyzing knowledge graph embedding methods from a multi-embedding interaction perspective," *arXiv*, 2020.

[7] L. Costabello *et al.*, "AmpliGraph: a Library for Representation Learning on Knowledge Graphs," Mar. 2019.

[8] A. Bordes *et al.*, "Translating Embeddings for Modeling Multi-relational Data," in *Neural Information Processing Systems (NIPS)*, South Lake Tahoe, United States, Dec. 2013, pp. 1–9.

[9] B. Yang *et al.*, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv*, 2015.

[10] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Phung, "A novel embedding model for knowledge base completion based on convolutional neural network," *Proceedings of the 2018 Conference of the NAACL-HLT, Vol. 2*, 2018.

[11] G. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[12] K. Bollacker *et al.*, "Freebase: A collaboratively created graph database for structuring human knowledge." ACM, 2008.

[13] F. Mahdisoltani, J. Biega, and F. M. Suchanek, "Yago3: A knowledge base from multilingual wikipedias," in *CIDR*, 2015.

[14] A. Rossi *et al.*, "Knowledge graph embedding for link prediction: A comparative analysis," vol. 15, no. 2, 2021.

[15] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[16] S. Kumar, X. Zhang, and J. Leskovec, "Learning dynamic embeddings from temporal interactions," *arXiv*, 2018.

[17] M. Wang, L. Qiu, and X. Wang, "A survey on knowledge graph embeddings for link prediction," *Symmetry*, vol. 13, p. 485, 03 2021.

[18] L. Cai and W. Y. Wang, "Kbgan: Adversarial learning for knowledge graph embeddings," *arXiv*, 2018.

[19] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[20] J. Xu, P. Venkatesh, and H.-J. Tsai, *Maximize TensorFlow* Performance on CPU: Considerations and Recommendations*. Intel, 2021.