On the Computational Cost of Knowledge Graph Embeddings

Victor Charpenay
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Mansour Zoubeirou A Mayaki
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Antoine Zimmermann
Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

Abstract

Over a decade, numerous Knowledge Graph Embedding (KGE) models have been designed and evaluated on reference datasets, always with increasing performance. In this paper, we re-evaluate these models with respect to their computational *efficiency* during training, by estimating the computational cost of the procedure expressed in floating-point operations. We design a cost model based on analytical expressions and apply it on a collection of 20 KGE models, representative of the state-of-the-art.

We show that dimensionality or parameter efficiency, used in the literature to compare models with each other, are not suitable to evaluate the true cost of models. Through fixed-budget experiments, a novel approach to evaluate KGE models based on cost estimates, we re-assess the relative performance of model families compared to the state-of-the-art. Bilinear models such as ComplEx underperform with a low computational budget while hyperbolic linear models appear to offer no particular benefit compared to simpler Euclidian models, especially the MuRE model. Neural models, such as ConvE or CompGCN, achieve reasonable performance in the literature but their high computational cost appears unnecessary when compared with other models. The trade-off between efficiency and expressivity of both linear and neural models is to be further explored.

2012 ACM Subject Classification Information systems → Resource Description Framework (RDF); Computing methodologies → Semantic networks; Hardware → Impact on the environment

Keywords and phrases Knowledge Graph Embedding, Parameter Efficiency, Computational Budget, Green AI

Digital Object Identifier 10.4230/TGDK.4.1.1

Category Research

Supplementary Material

Software: `https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments/` [14]
archived at `swh:1:dir:29d727e535031eca0583c489c67c47d0b9165d30`

Funding This work was supported by the European Lighthouse to Manifest Trustworthy and Green AI (ENFIELD) project funded by the European Union's HORIZON Research and Innovation Programme. Grant agreement No: 101120657.

Acknowledgements Computations have been performed on the supercomputer facilities of the Mésocentre Clermont-Auvergne of the Université Clermont Auvergne.

Received 2024-10-07 **Accepted** 2026-01-05 **Published** 2026-03-31

1 Introduction

In 2018, OpenAI researchers released a meta-analysis suggesting that the amount of compute to train state-of-the-art deep learning models had grown exponentially since 2012 [4]. This trend was confirmed a few years later by the coming of *large* language models such as GPT-3 [17], BLOOM [36] or LLaMa [43], which achieved unprecedented performance in natural language processing tasks.

This exponential trend has raised concerns about the sustainability of Artificial Intelligence (AI) research. It was estimated that training GPT-3 emitted as much CO_2 as 100 humans for their food, material goods, travel, etc. over a year (502 tCO_2e vs. 5 tCO_2e/capita/year[1]) [27]. Given that humanity's high carbon emissions are themselves not sustainable, the carbon intensity of GPT-3's training is clearly too high. BLOOM, an open-source model of comparable size, is 20 times less carbon intensive according to the same estimate (25 tCO_2e). LLaMa, following the same design principles as GPT-3 and BLOOM, has been shown to perform better with less trainable parameters. However, training LLaMa (65B parameters) required as much energy as training BLOOM (176B parameters), partly because of additional normalization layers and more complex activation functions, and emitted significantly more CO_2 (173 tCO_2e).

In the emerging field of Green AI, researchers aim to study the non-trivial relationship between the performance, parameter efficiency, computational cost and environmental impact of AI models [38]. Green AI models must not only be accurate at the task at hand, they must also be efficient: models must minimize compute to lower their environmental impact. As illustrated with BLOOM and LLaMa, parameter efficiency is not a good proxy to estimate the environmental impact of training a model. On the other hand, CO_2 emissions may greatly vary for the same training procedure depending on external conditions: for example, if BLOOM had been trained in the USA instead of France (whose energy mix includes 70% of nuclear energy), it would have emitted as much CO_2 as LLaMa. As a consequence, Green AI advocates encourage to report computational costs measured in floating-point operations (FLOPs[2]) instead. The objective of this paper is to study the computational cost of a specific class of models: Knowledge Graph Embedding (KGE) models.

KGE models are to Knowledge Graphs (KGs) what language models are to text corpora. They consist in vector representations of KG entities and relations that are meant to capture latent structures of an input KG (such as soft rules or implicit classes of entities). Through operations on vectors, models can generalize beyond what is explicitly stated in the KG and detect missing or erroneous information.

KGE models generally have a significantly lower cost than language models, even for large KGs. However, they seem to follow the same pattern of achieving better performance through more parameters and higher computational costs. TransE, one of the first KGE models published in 2013, was evaluated with 20 to 50 latent dimensions [11]. Six years later, the RotatE model was compared to TransE with 1,000 dimensions – with substantial absolute improvements for both models [40]. As we will see later in the paper, we estimate that to train RotatE in 20 times more dimensions required 3,000 times more compute than the original TransE model.

Morever, non-parametric models such as AnyBURL [28] and its successor, SAFRAN [32], meet the performance of embedding-based models with far less compute. Their approach consists in mining composition rules from the KG instead of learning latent embeddings. The authors of AnyBURL give the following comparison: after 1,000 s of mining on a single CPU, AnyBURL yields good results while the best performing KGE model on the same dataset, ComplEx-N3 [22],

[1] The estimate for human emissions is taken from Strubell *et al.* [39].
[2] Note that 1 FLOP is not the same as 1 FLOPS, which measures floating-point operations *per second*.

requires between 10,000 s and 50,000 s of training on a GPU. The major drawback of AnyBURL and SAFRAN, however, is that a considerable amount of compute is necessary in the inference phase, while operations in the TransE, RotatE or ComplEx-N3 latent spaces are relatively cheap once trained.

More recent publications question the need for high dimensions to train KGE models [6, 13]. The AttH model has been shown to perform well with as few as 32 dimensions [13]. However, if one goes beyond dimensionality, as encouraged in Green AI, it can be shown that AttH requires significantly more FLOPs than RotatE, itself requiring more FLOPs than TransE with the same number of dimensions. In this paper, we review and compare the efficiency of representative KGE models in order to identify those able to improve performance with minimal computational overhead and to guide future work towards KGE training procedures with the lowest possible computational cost. Several large-scale studies have already compared the efficiency of KGE models under various constraints, such as dimensionality [35, 2] and execution time [34], but to the best of our knowledge, no comparison with respect to computational cost, measured in FLOPs, has ever been done. We aim to fill this gap in the paper.

We start our study with an overview of state-of-the-art KGE models in Section 2. We then compare models with respect to their size (number of trainable parameters) in Section 3, to highlight the limitations of this measure. To evaluate the computational cost of KGE models in FLOPs, the preferred Green AI metric, we proceed in two steps: we provide estimates for inference cost in Section 4.2 and build up on these results to provide estimates for full training cost in Section 4.3. Training is indeed roughly equivalent to a sequence of inferences on positive and negative examples of KG statements. Finally, in Section 5, we consider fixed-budget performance obtained by setting an upper bound to the number of FLOPs allowed for training. Fixed-budget performance is not only relevant in contexts where the computing platform has low power supply, they also provide a fairer comparison point as we will see towards the end of the paper.

2 State-of-the-Art Models

KGE models have been designed for various inductive learning tasks: link prediction, entity classification, triple classification, query answering, entity similarity measure, etc [21, Sec. 5.2]. In this paper, we restrict ourselves to the most common task, link prediction in a transductive setting. Given a KG defined as a set of $\langle h, r, t\rangle$ triples, where $h, t \in E$ are referred to as (head and tail) entities and $r \in R$ as a relation, link prediction consists in evaluating the plausibility of an unknown triple $\langle h, r, t'\rangle$ or $\langle h', r, t\rangle$, in which h', t' are known to be in E. What a model must learn is then a scoring function $f : E \times R \times E \to \mathbb{R}$ such that $f(h, r, t)$ encodes the plausibility of triple $\langle h, r, t\rangle$. Note that $f(h, r, t)$ is not necessarily a probability, as it is not restricted to the interval $[0, 1]$. In practice, scoring is only used to rank triples.

Early KGE models were being evaluated on two datasets, FB15k and WN18, respectively derived from Freebase and WordNet [11]. FB15k includes about 15,000 entities and more than 1,000 relations. WN18 has more entites (∼40,000) but less relations (18). It has later been found that naive models, that look for pairs of inverse relations $(r, r^-$ such that $\langle h, r, t\rangle$ and $\langle t, r^-, h\rangle$ are in the dataset), perform well on both datasets [42, 18]. For a more challenging evaluation, the FB15k-237 and WN18RR variants were introduced, in which one of r or r^- was filtered out. A leaderboard reports results for at least 68 KGE models on FB15k-237[3] and on WN18RR[4]. Some of these models were also evaluated on larger datasets such as YAGO3-10 but to be able to compare all models with each other, we choose to focus on FB15k-237 and WN18RR only in this paper.

[3] https://paperswithcode.com/sota/link-prediction-on-fb15k-237
[4] https://paperswithcode.com/sota/link-prediction-on-wn18rr

Table 1 Knowledge Graph Embedding models selected for this study.

Linear		Bilinear	Neural
Euclidian	**Hyperbolic**		
TransE [11]	MuRP [8]	DistMult [49]	R-GCN [37]
RotatE [40]	AttH [13]	ComplEx [44]	ConvE [18]
BoxE [1]	ConE [6]	ComplEx-N3 [22]	ConvKB [29]
ExpressivE [33]		TuckER [9]	CompGCN [45]
MuRE [8]		QuatE [50]	NBFNet [52]
AttE [13]		DualE [12]	

Most recent models, including LP-BERT [23], MoCoSA [20], Relphormer [10] and KERMIT [24], rely on pre-trained language models for learning. The computational cost of training language models is an order of magnitude higher than that of KGE models. The base version of BERT, for instance, has 110M trainable parameters [19] whereas the KGE models we consider here have up to 40M parameters. Its FLOP count is also significantly higher. For inference, BERT requires 131M FLOPs to output a single token (see calculation details in Appendix A). This estimate is a lower bound, given that a triple is represented at least with three tokens. In comparison, the two most expensive models in our study require 83M and 16M FLOPs to score the entire triple. In this paper, we choose to focus on KGE models that take a KG as the only input to training. Such models can only learn structural features of the KG (as opposed to textual features, in the case of pre-trained language models).

Out of the numerous KGE models available in the literature, we made a selection of 20 models[5]. These models are listed in Table 1. KGE models have traditionally been classified in three main categories: linear or distance-based models, bilinear or tensor factorization models and neural models [30, 47, 21]. Linear models can be subdivided into Euclidian models and hyperbolic models. TransE and RotatE are Euclidian models while AttH (which performs well in low dimensions but requires more FLOPs than RotatE) is a hyperbolic model, for instance. As the distinction between the two may be important in a computational cost analysis, we selected models such that both Euclidian and hyperbolic categories, as well as bilinear and neural categories, are evenly represented. We selected up to 5 models in each category: 4 Euclidian models, 3 hyperbolic models, 5 bilinear models and 5 neural models. Two out of the 3 hyperbolic models have Euclidian variants that have also been evaluated on FB15k-237 and WN18RR. One of the bilinear models, ComplEx [44], also has a variant that shows better performance. For comparison purposes, we included these 4 additional models, leading to a total of 20 KGE models. We describe each category of models in the remainder of the section.

2.1 Euclidian Models

Linear (Euclidian and hyperbolic) models have in common that their scoring function is defined in terms of a distance between head and tail entity embeddings. Most linear scoring functions have the following generic form:

$$f(h, r, t) = -\|g_r(\mathbf{e}_h) - \mathbf{e}_t\| \tag{1}$$

[5] These models were published between 2013 and 2021. Best-performing models published after 2021, from KG-BERT to KERMIT, were excluded due to the fact they rely on pre-trained language models.

where $\mathbf{e}_x$ is a vector embedding of term x in $\mathbb{R}^d$, $\mathbb{C}^d$ and $\|\cdot\|$ is the L1-norm or L2-norm. Embedding $\mathbf{e}_h$ is first transformed by a function specific to relation r, then compared to $\mathbf{e}_t$. The closer the two embeddings are, the more likely it is that triple $\langle h, r, t\rangle$ belongs to the KG.

The simplest model, TransE, represents relations as a single vector $\mathbf{w}_r \in \mathbb{R}^d$ used as a translation offset to transform $\mathbf{e}_h$. Scoring is done by calculating $f_{\text{TransE}}(h, r, t) = -\|(\mathbf{e}_h + \mathbf{w}_r) - \mathbf{e}_t\|$. RotatE and AttE replace translation with rotation, defined by the following block-diagonal matrix:

$$\mathbf{W}_r = \begin{bmatrix} \mathbf{R}_{\theta_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{R}_{\theta_d} \end{bmatrix} \tag{2}$$

where $\mathbf{R}_{\theta_i}$ is a 2D rotation matrix:

$$\mathbf{R}_{\theta_i} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{bmatrix} \tag{3}$$

Relation embeddings can be concisely represented as a rotation vector $\theta \in \mathbb{R}^d$ but they can also be represented as a complex vector $\mathbf{w}_r \in \mathbb{C}^d$. If every member $w_{r,i}$ of $\mathbf{w}_r$ is normalized such that $w_{r,i} = \cos(\theta_{r,i}) + i\sin(\theta_{r,i})$, the two representations are equivalent. RotatE uses the complex representation and defines scoring as $f_{\text{RotatE}}(h, r, t) = -\|\mathbf{e}_h \odot \mathbf{w}_r - \mathbf{e}_t\|$, where $\odot$ is the Hadamard product. AttE uses the former representation.

Along with translation and/or rotation, some models introduce scaling. For instance, MuRE combines scaling and translation, giving the scoring function $f_{\text{MuRE}}(h, r, t) = -\|(\mathbf{e}_h \odot \mathbf{w}_r + \mathbf{w}'_r) - \mathbf{e}_t\|$. Contrary to RotatE, MuRE embeddings are defined over $\mathbb{R}^d$. AttE distinguishes itself from other linear models by introducing a further geometric transformation: reflection. However, as rotation and reflection are not entirely independent, AttE also introduces an attention mechanism across rotation and reflection with more parameters to train, i.e. with a higher computational cost. BoxE and ExpressivE are both translational but they have another distinguishing feature: they introduce a width parameter in each dimension in order to view relation embeddings as regions instead of points in the latent space. This also comes at the price of a higher computational cost.

A number of other linear models, such as TransR [26] and HAKE [51], also apply geometric transformations to the head entity before scoring. These models are not included in our study.

2.2 Hyperbolic Models

Hyperbolic linear models – MuRP, AttH and ConE in our study – redefine rotation, translation, scaling and reflection. Contrary to classical Euclidian space, distance in a hyperbolic space of curvature c is given by the following formula:

$$d_c(\mathbf{x}, \mathbf{y}) = \frac{2}{\sqrt{c}} \text{arctanh}(\sqrt{c}\|\mathbf{x} \oplus^c \mathbf{y}\|) \tag{4}$$

where $\oplus^c$ is the Möbius addition (the hyperbolic translation operator), defined as follows:

$$\mathbf{x} \oplus^c \mathbf{y} = \frac{(1 + 2c\mathbf{x}^T\mathbf{y} + c\|\mathbf{x}\|^2)\mathbf{x} + (1 - c\|\mathbf{x}\|^2)\mathbf{y}}{1 + 2c\mathbf{x}^T\mathbf{y} + c^2\|\mathbf{x}\|^2\|\mathbf{y}\|^2} \tag{5}$$

Scoring in hyperbolic KGE models is performed by calculating $-d_c(g_r(\mathbf{e}_h), \mathbf{e}_t)$ instead of $-\|g_r(\mathbf{e}_h) - \mathbf{e}_t\|$.

MuRP and AttH are equivalent to MuRE and AttE, up to a redefinition of operators in hyperbolic space. MuRP is a combination of translation and scaling while AttH uses rotation and reflection (with some attention weight). ConE combines rotation and scaling but, to the best of our knowledge, no Euclidian counterpart to ConE exists in the literature.

Among other properties, geometries in hyperbolic spaces are known to efficiently capture hierarchies of entities: a tree of entities can be embedded in a 2D hyperbolic space such that all entities at level i are equally distant to their parent at level $i-1$, without constraing the respective distance between same-level entities. The distance between embeddings of entities e_1 and e_2 can then represent the depth of e_2 in the hierarchy relative to e_1 while the distance between e_1 and e_3, at the same level, may represent their similarity for some non-hierarchical relation.

Other non-Euclidian, non-linear models also adapt the idea of defining a distance between head and tail entity representations. Bayesian models, for instance, measure a distance between probability distributions [46]. Such models are not included in our study.

2.3 Bilinear Models

Bilinear models define scoring in terms of a dot product instead of a distance, as follows:

$$f(h,r,t) = Re(g_r(\mathbf{e}_h) \cdot \overline{\mathbf{e}_t}) \tag{6}$$

where $Re(x)$ is the real part of complex number x and $\overline{\mathbf{x}}$ is the conjungate of complex vector $\mathbf{x}$. If $\mathbf{x} \in \mathbb{R}^d$, then $\mathbf{x}$ and $\overline{\mathbf{x}}$ are identical.

DistMult and ComplEx both define g_r as the Hadamard product with some relation embedding $\mathbf{w}_r$ but ComplEx gains expressivity compared to DistMult by simply going from real-valued to complex-valued embedding. The scoring function of ComplEx is $f_{\text{ComplEx}}(h,r,t) = \text{Re}((\mathbf{e}_h \odot \mathbf{w}_r) \cdot \overline{\mathbf{e}_t})$. QuatE and DualE look for more generality by extending further to quaternion-valued and dual quaternion-valued embeddings. Members of quaternion space $\mathbb{H}$ are generalizations of complex numbers of the form $a + ib + jc + kd$, where i, j and k are imaginary numbers. Members of the dual quaternion space have twice as many components. Both spaces redefine addition and multiplication, requiring more FLOPs than their scalar counterpart.

TuckER is an attempt to redefine and generalize ComplEx in $\mathbb{R}^d$. It is based on the idea that KGs, if seen as tensors of order 3, can be factorized into entity- and relation-specific embeddings, respectively in $\mathbb{R}^{d_e}$ and $\mathbb{R}^{d_r}$. To recover the full tensor, these factors can be multiplied to a core tensor defined in $\mathbb{R}^{d_e \times d_r \times d_e}$. This generalization involves significantly more computation than ComplEx, which might explain why ComplEx and ComplEx-N3 have remained the most popular KGE models so far. MetaSD [25], for instance, derives from ComplEx. It performs distillation on a pre-trained ComplEx model, intending both to reduce the size of the original model and to improve its performance. Model distillation goes beyond the scope of our study, though.

Other bilinear models include RESCAL [31], the very first wide-spread KGE model, and HypER [7], which was inspired by ConvE (see next section) and led to the design of TuckER.

2.4 Neural Models

The last category is slightly more heterogeneous. It includes various neural models whose architecture features 1D convolution (ConvKB), 2D convolution (ConvE) or graph convolution (R-GCN) and other message passing operations. A distinctive feature of these models is the introduction of non-linearities, such as the ReLU function, to latent vectors before calculating a score.

R-GCN, CompGCN and NBFNet follow the general architecture of Graph Neural Networks (GNNs), which consists in a sequence of combine-aggregate operations that follow the structure of the KG: the embedding of an input entity is calculated from the embeddings of its neighbors, in 2 steps. In the combine step, the embeddings of all neighbors are transformed by a relation-specific function. In the aggregate step, those intermediary embeddings are aggregated into a single vector

that can be used as a representation for the input entity. Several combine-aggregate operations can be computed on top of each other, forming GNN layers. The formulation for R-GCN is the following:

$$\mathbf{h}_t^{(l+1)} = \mathrm{ReLU}\left(\sum_{r\in R}\sum_{h\in B_{t,r}} \frac{1}{|B_{t,r}|}\mathbf{W}_r^{(l)}\mathbf{h}_h^{(l)}\right) \tag{7}$$

where $B_{t,r}$ is the set of neighbors of t ($h \in B_{t,r}$ if $\langle h, r, t\rangle$ is in the KG), $\mathbf{h}_e^{(l)}$ is the embedding of entity e at layer l and $\mathbf{W}_r^{(l)}$ is a trainable matrix for relation r at layer l. To reduce the number of trainable parameters per relation, $\mathbf{W}_r^{(l)}$ is defined as a block-diagonal matrix, which makes R-GCN close to bilinear models in spirit. CompGCN uses cross-correlation instead of matrix multiplication while NBFNet uses scaling and translation operators, and extends the calculation to all entities in the KG (as opposed to neighbors only).

Note that GNNs only produce embeddings. When training GNNs for link prediction, the obtained embeddings must still be compared to calculate a final score. R-GCN applies DistMult (with an additional relation embedding), CompGCN uses a pre-trained ConvE model and NBFNet appends a dedicated multilayer perceptron to the network to reduce embeddings to a scalar. GNNs thus necessarily require more FLOPs than the simpler models they may rely on.

Neural models also include models based on the Transformer architecture, which has recently found many applications. LP-BERT, MoCoSA, Relphormer and KERMIT all fall into this category because they rely on BERT, a bidirectional Transformer encoder. Other models, including HittER [16], adapt the Transformer architecture to KG structures instead of turning triples into text. However, at the time of writing, such Transformer-based models are still outperformed by NBFNet on FB15k-237 and WN18RR, despite significant FLOP counts. We decided to leave them for future work.

2.5 Training Procedure

All scoring functions mentioned above are to be trained by stochastic gradient descent, an iterative process. At each step, the score of each KG triple is compared to the score of one or more triples not stated in the KG, yielding a loss value that is to be minimized. Various loss functions have been introduced to optimize scoring functions. The most common loss is binary cross-entropy, treating the task of link prediction as binary classification. However, several studies suggest that loss functions are not tightly coupled with scoring functions [2, 35]: several models have been shown to perform better if trained with loss functions not originally designed for these models.

Several of the losses found in the KGE literature, including cross-entropy loss, are subsumed by the following formula:

$$\mathcal{L}(\tau) = -\log(\sigma(\lambda + f(\tau))) - \sum_{\tau'\in N_\tau} w_{\tau'}\log(1-\sigma(\lambda + f(\tau'))) \tag{8}$$

where $\tau = \langle h, r, t\rangle, \tau' = \langle h', r', t'\rangle$ are triples, σ is the sigmoid function and N_τ is a set of "negative" triples, assumed to be false statements. The standard practice for choosing N_τ is to construct it from $\langle h, r, t\rangle$, setting $r = r'$ and either $h = h'$ or $t = t'$ but not both.

The formula for $\mathcal{L}$ has two hyperparameters: λ and $w_{\tau'}$ (for each $\tau' \in N_\tau$). The λ value is a margin that intuitively captures that above a certain score (the margin), triples may be considered true and false otherwise. This margin hyperparameter is especially important to train linear models: since they can only output negative scores, the sigmoid function would only output values below 0.5 without a margin, with small gradients. Bilinear models more commonly use standard cross-entropy and set $\lambda = 0$. The other hyperparameter, $w_{\tau'}$, is either set to $w_{\tau'} = 1/|N_\tau|$ or to

some attention weight that depends on $f(\tau')$. It was indeed observed that applying a variable attention weight to negative triples, to focus on non-trivial triples with high classification error, yielded significantly better results on linear models [40]. In this setting, weight $w_{\tau'}$ is defined as the i-the element of vector $\text{softmax}([f(\tau'_1); \ldots; f(\tau'_{|N_\tau|})])$, given $\tau'_i = \tau'$.

The loss formula we give here is defined for a single positive triple. Dettmers *et al.* first noted that the loss may also be defined for several positive triples at once, e.g. all triples of the form $\langle e, r, t_1 \rangle, \ldots, \langle e, r, t_n \rangle$ [18]. We do not consider this case in our study. Some KGE models also add regularization terms to $\mathcal{L}$. The most common example is L2 regularization, which encourages embeddings to be small by adding a squared norm to the loss, as follows:

$$\mathcal{L}'(\tau) = \mathcal{L}(\tau) + \alpha\|\theta\|^2 \tag{9}$$

where α is a hyperparameter and θ is the concatenation of all trainable parameters involved in the computation of $\mathcal{L}(\tau)$. N3 regularization is a variant specifically designed for KGEs, involving cubic norms. ComplEx and ComplEx-N3 only differ in the regularization they apply during training: ComplEx applies L2 regularization by default whereas ComplEx-N3 applies N3 regularization.

3 Parameter Efficiency

It is difficult to uniformly calculate the computational cost of KGE models. As our state-of-the-art review shows, scoring functions rely on different embedding spaces ($\mathbb{R}^d$, $\mathbb{C}^d$, or $\mathbb{H}^d$, the quaternion vector space), different geometric operators (translation, rotation, scaling, reflection or more general matrix operations) and different distance functions (Euclidian or hyperbolic). Atomic operations underlying these components have no agreed-upon FLOP estimates. The only measure that is consistently reported across publications is d, the number of dimensions per entity or relation embedding. Yet, this number hides important details of the computation of plausibility scores. For instance, for the same hyperparameter d, RotatE and ComplEx would have twice as many parameters as TransE or DistMult, because they are defined in a complex embedding space, and thus require at least twice as many multiply-add operations. Because dimensionality is a poor proxy for computational cost overall, some publications also report *parameter efficiency*, defined as the minimum number of trainable parameters required for a model to reach a given performance. ConvE was shown to be 17 times more parameter-efficient than R-GCN and 8 times more parameter-efficient than DistMult for a given performance [18]. Before trying to express computational cost estimates with respect to FLOPs, we first study the parameter efficiency of KGE models.

3.1 Approach

Our objective in this preliminary study is to compare the performance of models with respect to their size. For each model, we expressed their size as an analytical expression counting their total number of parameters, as shown in Table 2. This expression depends on variables such as d (number of latent dimensions of the model) and $|E|$, $|R|$ (number of entities and relations in the KG). For instance, a TransE or DistMult model has d parameters for each entity and for each relation, giving a size of $d(|E| + |R|)$. Most non-neural KGE models only depend on these variables. The only exception is TuckER, which can have distinct embedding sizes for entities and relations: d becomes d_e for entities and d_r for relations. Neural models have more variability. Convolutional architectures (ConvKB, ConvE) also have a variable number of convolution filters denoted c and a filter size s_k. GNNs architectures typically have more than one layer. Their total number of layers is denoted L while their number of hidden layers (often with a different structure than input and output layers) is denoted m. Hidden layers have d_h dimensions. R-GCN additionally depends on b, the size of each block in block-diagonal aggregation matrices.

Table 2 Size of Knowledge Graph Embedding models ($|E|$: nb. of entities, $|R|$: nb. of relations, d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, θ_x: parameters of model x).

Model	Nb. parameters
TransE	$d(\|E\| + \|R\|)$
RotatE	$2d(\|E\| + \|R\|)$
BoxE	$d(2\|E\| + 4\|R\|)$
ExpressivE	$d(\|E\| + 6\|R\|)$
MuRE	$d(\|E\| + 2\|R\|) + \|E\|$
AttE	$d(\|E\| + 3\|R\|) + \|E\|$
MuRP	$d(\|E\| + 2\|R\|) + \|E\|$
AttH	$d(\|E\| + 3\|R\|) + \|E\| + \|R\|$
ConE	$2(d+1)\|E\| + 3d\|R\|$
DistMult	$d(\|E\| + \|R\|)$
ComplEx	$2d(\|E\| + \|R\|)$
ComplEx-N3	$2d(\|E\| + \|R\|)$
TuckER	$d_e^2 d_r + d_e\|E\| + d_r\|R\|$
QuatE	$4d(\|E\| + \|R\|)$
DualE	$8d(\|E\| + \|R\|)$
R-GCN	$d(\|E\| + bL(\|R\| + 1)) + d\|R\|$
ConvE	$d(\|E\| + \|R\|) + 2cd(d+3) + cs_k + 7d$
ConvKB	$d(\|E\| + \|R\|) + (4+d)c$
CompGCN	$d(\|E\| + \|R\|) + 4dd_h + \|\theta_{\text{ConvE}}\|$
NBFNet	$d\|R\|(Ld + L + 1) + Ld(13d+3) + d_h(2d+1) + d_h + 1$

Only few KGE models have several possible formalizations, hence possibly ambiguous parameter counts: R-GCN, RotatE, DualE and ConE. R-GCN has two variants, one that has been tested on entity classification, the other on link prediction. We only consider the link prediction variant (the one involving block-diagonal matrices) in this paper. In addition, for brevity, the R-GCN formula covers in fact a special case in which all layers have the same size, which is not mandatory in GNN architectures. The parameter counts we give here intend to mirror concrete model implementations, which are not necessarily optimal counts. RotatE and DualE respectively have $2d$ and $8d$ parameters per relation embedding but these parameters are in fact constrained, such that there are only d and $6d$ independent parameters per embedding. In the case of RotatE, the real and imaginary parts of each complex number could be reduce to a single rotation angle. The concrete implementation of RotatE still uses $2d$ parameters for each relation because rotation defined as a Hadamard product is more numerically stable (and faster) than trigonometric calculation. The same applies to DualE, in dual quaternion space. Finally, ConE may also have less parameters if KG relations were labeled as hierarchical or non-hierarchical (by an external source). Scoring indeed differs for the two types of relations. For fair comparison with other models, which are supposed to capture properties of relations through training, we chose the most general form for ConE involving a weighted sum of the two scoring functions.

Previous work already inspected the parameter efficiency of some of the models. Pavlović and Sallinger [33], and Zhu *et al.* [52] provide formulas for some of the models, which we report without changes. Other authors sometimes provide space complexity formulas [1, 13, 45]. These formulas are not accurate estimates, as they ignore constant factors. BoxE, for instance, has the

same space complexity as TransE or ExpressivE but it actually has twice as many parameters for entity embeddings. Some other authors also report parameter counts for given d, d_e and d_r values on FB15k-237 and WN18RR [18, 50, 9]. We made sure that our formulas are consistent with these counts for the two datasets.

Analytical expressions given in Table 2 are only the first step towards an evaluation of parameter efficiency. In a second step, we collected performance evalations for all models in the literature. That is, we collected hyperparameter values for d, b, c and other variables that authors chose in their experiments, together with the reported performance for their models. With this information, we can compare performance and size of a model for each experiment, where the size of the model is calculated from the corresponding analytical expression. For instance, Nguyen *et al.* evaluated TransE on FB15k-237 with $d = 100$ and reported a performance of hits@10 = 0.465 [29]. For FB15k-237, we have $|E| = 14,541$ and $|R| = 237$. The size of the TransE model in this case is $|\theta_{\text{TransE}}| = d(|E| + |R|) = 100 \times (14,541 + 237) = 1,477,800$. In another experiment, Sun *et al.* evaluated RotatE with $d = 500$ on WN18RR and reported a performance of hits@10 = 0.571 [40]. For WN18RR, we have $|E| = 40,943$ and $|R| = 11$, such that $|\theta_{\text{RotatE}}| = 2d(|E| + |R|) = 2 \times 500 \times (40,943 + 11) = 40,954,000$.

We found 26 distinct link prediction experiments for FB15k-237 and 23 experiments for WN18RR. There were two different settings for TransE and ComplEx on FB15k-237 and none for R-GCN on WN18RR. Hyperbolic models had two different settings on both datasets except ConE. All other models have a single experimental setup for each dataset. ConvKB is an special case: initial evaluation results were invalidated by later third-party experiments that uncovered an issue in the link prediction experimental protocol [41]. We consider the latter only, with lower performance for the same number of parameters. The complete list of experiments we selected, with performance and hyperparameter values, is available in Appendix B.

Across experiments, the reported performance metrics are not always identical. Some papers report Mean Reciprocal Rank (MRR) while others report Mean Rank (MR), rarely both. Hits@k, which gives the ratio of test triples whose score rank is lower than k, are commonly used as a metric as well but k may vary across papers. We found hits@10 to be the most consistently used metric for link prediction. We limit our study, as well as subsequent cost studies, to this metric. It is also worth noting that $|R|$ is not always the same across experiments. In some papers, results are given for models with $2|R|$ relation embeddings, as per a common data augmentation technique [18, 22]: for every embedding capturing a latent representation of some relation r, there is a second embedding capturing a representation of its inverse relation r^-, artificially added to the input KG.

3.2 Performance-Size Comparison

Parameter counts vs. hits@10 of all found experiments are synthesized in Figure 1. At first sight, it appears that numerous models outperformed baselines at the expense of significantly more parameters. More than half of the models have 2 to 8 times more parameters than DistMult, ComplEx, ConvKB or TransE, for both FB15k-237 and WN18RR. The only exception is NBFNet, which performs significantly better than any other model with the same number of parameters. MuRE and MuRP improve over TransE but on WN18RR, both models have 3 times more parameters. These numbers are not necessarily optimal[6] but we assume that authors have put effort in finding reasonably good hyperparameters, which allows us to compare models with a given performance. For equal performance, AttE has 1.5 times less parameters than TuckER,

[6] Authors of MuRE/MuRP report results for 200-dimensional embeddings but they do not guarantee that smaller embeddings cannot reach the same performance (with longer training cycles or different hyperparameters).

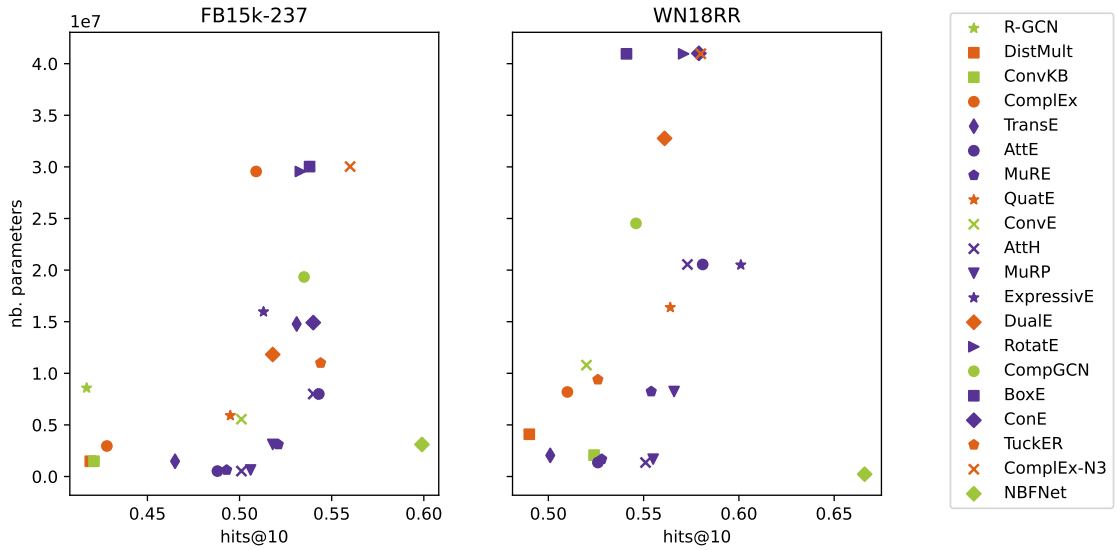


Figure 1 Model size with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

2 times less than ConE and 4 times less than BoxE, for example. All these models indeed reach hits@10 close to 0.54 on FB15k-237 – the models rank the correct target entity among top 10 candidates for 54% of test triples.

This also allows us to hypothesize that there exist more parameter-efficient, non-neural models than ComplEx-N3, which performs well but is among the largest models for both datasets. It is however unlikely that any linear or bilinear can reach the performance of NBFNet with a comparable number of parameters. Both TransE and ComplEx were evaluated twice on FB15k-237, with 10 times more dimensions in the second experiment than in the first but with an absolute increase in hits@10 of only 6.6% and 8.1%. NBFNet outperforms the largest TransE and ComplEx models by another margin of 6.8% and 9.0%. On WN18RR, ExpressivE obtains good results but requires more parameters than half of the models.

By adding model size as another comparison point, Figure 1 highlights the high heterogeneity of state-of-the-art KGE models. For comparable hits@10, differences in parameter counts can be as high as 1 to 4 (between AttE and BoxE). What the figure hides, however, is that under heterogeneous model architectures, spatial complexity is a poor proxy for computational cost. GPT-3, BLOOM and LLaMa could be more legitimately compared against their size because they all rely on the Transformer architecture. The size of latent embeddings and the number of attention heads in a Transformer, which yield a given parameter count, strongly influence the number of FLOPs necessary for inference with these models. This is not true for KGE models, especially when comparing Euclidian and hyperbolic models. MuRE and MuRP, on the one hand, and AttE and AttH, on the other hand, have the same number of parameters for a similar performance but hyperbolic operators (Möbius addition, hyperbolic distance) are computationally more intensive than their Euclidian counterpart. Euclidian models could therefore have more parameters and still be more efficient than hyperbolic models. In 32 dimensions, AttH was shown to outperform AttE (by a 2.5%-5% margin on FB15k-237 and WN18RR) but no comparison was made with a higher-dimensional AttE model, with equal computational cost. The same holds for MuRE and MuRP. In the following, we go beyond parameter efficiency and compare models with respect to their computational cost.

4 Computational Cost

As already mentioned, FLOPs are a relevant intermediary between easy-to-calculate but inaccurate model sizes and environmental impact criteria such as CO_2 emissions, which typically depend on external conditions. Estimating a number of FLOPs from a mathematical expression such as $f_{\text{TransE}}(h, r, t)$ should be independent of the underlying computing platform (CPU, GPU) and concrete implementations of higher-level operations (PyTorch, TensorFlow, etc.), in such a way that the resulting number can be compared with $f_{\text{ComplEx}}(h, r, t)$ and other expressions for the same KG. This requirement imposes some constraints on what 1 FLOP is but it does not alleviate all ambiguities.

Software utilities such as `thop`[7] or `ptflops`[8] can automatically count FLOPs from model implementations in PyTorch. These utilities mostly provide coarse-grained estimates for widely used Neural Network layers (convolution, ReLU, max pooling, etc.) but not for finer-grained linear or bilinear operations in use in KGE models. They also ignore transcendental functions such as square root or exponential, as did OpenAI researchers in their 2018 study of deep learning models [4]. These functions often occur in the formalization of KGE models. Furthermore, their estimates may depend on implementation details. As an example, the following PyTorch statement, often encountered:

```
(a - b).norm(dim=-1)**2
```

induces more computation but is mathematically equivalent to:

```
((a - b)**2).sum(dim=-1)
```

where the first statement computes the squared L2-norm $(\|\mathbf{a} - \mathbf{b}\|)^2$ while the second statement directly computes $\sum_d (a_i - b_i)^2$. Ideally, the same number of FLOPs should be derived from the 2 statements – a requirement that existing utilities do not meet. In our study, we define analytical expressions in a semi-manual fashion instead. Our methodology is explained next.

4.1 Approach

In their 2018 study, OpenAI researchers restrict the definition of a FLOP to add and multiply operations – a common practice. Expressions $a + b$ and $a \times b$, where $a, b \in \mathbb{R}$ both have a cost of 1 FLOP. On that basis, they apply the following formula to calculate the cost of training neural model x [4]:

$$\begin{aligned} \kappa_x = \quad & \text{number of add-multiplies per forward pass} \\ & \times \text{ 2 FLOPs per add-multiply} \\ & \times \text{ 3 (for forward and backward pass)} \\ & \times \text{ number of examples in dataset} \\ & \times \text{ number of epochs} \end{aligned} \tag{10}$$

Add and multiply operations are commonly implemented in a single sequence of low-level instructions, referred to as a Multiply-Accumulate (MAC) operation, to limit rounding errors. However, a single MAC counts as 2 FLOPs. The FLOP count for a single inference step (or forward pass) is then multiplied by the number of examples on which the model is trained and a number of epochs. In our notation, the number of training examples is given by $|T|(1 + |N_\tau|)$, where T is the set of training triples in the KG.

[7] `https://github.com/Lyken17/pytorch-OpCounter/`
[8] `https://github.com/sovrasov/flops-counter.pytorch`

We take Definition 10 as a starting point, though ignoring the factor 3 accounting for back-propagation. This factor is indeed important when correlating FLOP count with training time on a CPU or GPU platform but in our case, we only intend to compare FLOP counts with each other. We then develop design principles to overcome the limitations of existing FLOP counting utilities: count of operations beyond MAC operations, especially transcendental function evaluation, on the one hand and algebraic reduction of equivalent operations on the other hand.

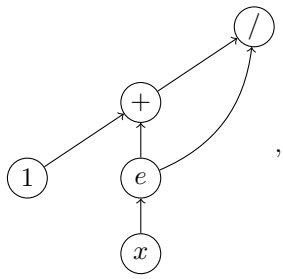
We adopt a compositional approach to FLOP counting, according to the following principles:

1. every mathematical operation on real values, including transcendental function application, costs 1 FLOP;
2. the cost of non-transcendental functions and functions on non-real values, is evaluated on their simplest analytical expression (the one with the lowest cost);
3. if a sub-expression appears in several places in the evaluated formula, it is counted only once.

From these principles, we define a FLOP count for basic expressions on scalars, vectors and matrices. The FLOP count of a given scoring function is then obtained by summing the count of all sub-expressions. It is important to keep in mind that resulting FLOP counts should not be taken as a basis to estimate execution times, which heavily depend on implementation details and on the available hardware. For instance, the function `arctanh`, rarely used in practice, has a slow PyTorch implementation on CPU. On the contrary, recurrent expressions such as $\log(1+x)$ have a dedicated, faster function in PyTorch (`log1p`). The correspondance between FLOP count and execution time is further explored in Appendix C.

We illustrate our approach with the sigmoid function. Its analytical expression, $\sigma(x) = e^x/(e^x+1)$, can be represented as a graph $G_{\sigma(x)}$ whose vertices are sub-expressions labeled either with an operator or an atomic operand. If a sub-expression appears n times in the expression, it has n outgoing edges to other vertices. Each vertex is given a FLOP count, depending on its operator and the dimension of its operands. The total cost $\kappa_{\sigma(x)}$ is the sum of FLOP counts of all vertices in $G_{\sigma(x)}$, as illustrated below:

$$G_{\sigma(x)} = \quad , \quad V_{\sigma(x)} = \{\textcircled{1}, \textcircled{x}, \textcircled{e}, \textcircled{+}, \textcircled{/}\}, \quad \kappa_{\sigma(x)} = \sum_{v \in V_{\sigma(x)}} \kappa_v \quad (11)$$



The cost of each operation $(e, +, /)$ is 1 FLOP and the cost of operands $(1, x)$ is 0 FLOP, which gives $\kappa_{\sigma(x)} = 3$. The sigmoid function can equivalently be expressed as $\sigma(x) = 1/(1+e^{-x})$, which however would give a cost of 4. In this work, we aim to provide a lower bound for FLOPs and thus keep the minimal value $\kappa_{\sigma(x)} = 3$. In addition, if the input is a d-dimensional vector instead of a scalar, the sigmoid function is mapped to each dimension, giving $\kappa_{\sigma(\mathbf{x})} = 3d$.

Table 3 summarizes FLOP counts of common KGE scoring expressions. The cost of vector operations is derived from costs on real values, accounting for the fact that sum reduction is fused with multiplication into MAC operations. For instance, the dot product of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ requires d multiplications ($x_i y_i$ for $1 \leq i \leq d$) and $d-1$ additions but we count instead d MAC operations, i.e. $2d$ FLOPs. Instead of making MACs explicit in cost calculation, we define that sum reduction of vector $\mathbf{x} \in \mathbb{R}^d$ costs d FLOPs. As a consequence, calculating the L1-norm of vector $\mathbf{x}$ has the same cost as a dot product: it requires d absolute value calculations and a sum reduction of d FLOPs. The L2-norm requires one more operation (square root). If a mathematical

Table 3 Computational cost of common operations.

	Operation	Cost
$x, y \in \mathbb{R}$	$x+y, xy, x-y, x/y, \lvert x\rvert, \max(x,y)$	1
	$\sqrt{x}, \log(x), e^x, \tanh(x), \text{arc}^*(x)$	1
	$\sigma(x) = e^x/(e^x+1)$	3
$x, y \in \mathbb{C}$	$\overline{x}$	1
	$x+y$	2
	$\lvert x\rvert$	4
	xy	6
$\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$	$\lVert\mathbf{x}\rVert_1, \mathbf{xy}$	$2d$
	$\lVert\mathbf{x}\rVert_2$	$2d+1$
	$\mathbf{x} \oplus^1 \mathbf{y}$	$16d+11$
	$\mathbf{x} \oplus^c \mathbf{y}, c \neq 1$	$16d+15$
	$d_c(\mathbf{x}, \mathbf{y})$	$19d+22$
$\mathbf{x} \in \mathbb{C}^d$	$\lVert\mathbf{x}\rVert_2$	$4d+1$
	$\lVert\mathbf{x}\rVert_1$	$5d$
$\mathbf{A} \in \mathbb{R}^{d_1 \times d_2}, x \in \mathbb{R}^{d_1}, y \in \mathbb{R}^{d_2}$	$\mathbf{xA}, \mathbf{Ay}$	$2d_1d_2$

expression includes $\|\mathbf{x}\|_2^2$, instead of giving it a cost of $2d+2$ FLOPs (cost of L2-norm plus cost of multiplication), we first reduce it to $\sum_d x_i^2$ and obtain a cost of $2d$ FLOPs. Note that in our cost model, comparison operators also cost 1 FLOP. $|x|$ and $\max(x,y)$ both cost 1 FLOP and since they are equivalent to the ternary operations $x > 0\ ?\ x : -x$ and $x > y\ ?\ x : y$, we consider the comparison to also have a cost of 1 FLOP for consistency. This further extends the usual definition of FLOP (which only considers add and multiply operations).

Operations on complex numbers are derived from operations on their real and imaginary parts. The addition of $x = a + ib$ and $y = a' + ib'$ is the complex number $x + y = (a + a') + i(b + b')$, requiring 2 additions ($a + a'$ and $b + b'$). Their multiplication is $xy = (aa' - bb') + i(ab' + ba')$, requiring 4 multiplications (aa', bb', ab' and ba'), 1 addition ($ab'+ba'$) and 1 substraction ($aa'-bb'$), for a total cost of 6 FLOPs. Similarly, addition and multiplication in $\mathbb{H}$ have a cost in FLOPs of 4 (4 additions) and 28 (4×4 multiplications and 3×4 additions). In complex and quaternion spaces, the absolute value – or norm – of a single number is defined as the Euclidian norm of their components, with a cost of 4 and 8 FLOPs respectively. When computing the Euclidian norm of a complex vector $\mathbf{x} \in \mathbb{C}^d$, it is not necessary to compute the full absolute value of its elements x_i ($1 \leq i \leq d$). The entire computation reduces to summing the square of the real and imaginary components of all elements ($2d$ FLOPs for squares, $2d$ FLOPs for sum reduction), followed by computing a square root, for a total cost of $4d + 1$ FLOPs.

Following the principles detailed above, we are able to obtain an analytical formula for the total cost of scoring for each KGE model. The result is in Table 4. In the scoring function of ComplEx, we considered the lowest possible amount of compute: since only the real part of the bilinear product is considered, scoring requires to compute the sum of four operands of the form $\mathbf{x} \odot \mathbf{y} \cdot \mathbf{z}$, where $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^{d/2}$. The total cost of ComplEx scoring is then $4 \times 3d$ per operand plus 3 FLOPs for the final sum. The scoring function of BoxE and ExpressivE includes a choice operator $\mathbf{b}\ ?\ \mathbf{x} : \mathbf{y}$, such that the output vector at index i is x_i if boolean value b_i is true or y_i otherwise. We consider that evaluating this operator costs as much as computing the values of $\mathbf{b}$, $\mathbf{x}$ and $\mathbf{y}$, which matches GPU implementations (based on vectorization), though typical CPU

Table 4 Computational cost of Knowledge Graph Embedding models (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, S: batch size, κ_x: computational cost of model x).

Model	Nb. operations (score)
TransE	$4d + 1$
RotatE	$10d + 1$
BoxE	$44d + 2$
ExpressivE	$34d + 1$
MuRE	$5d + 3$
AttE	$29d + 13$
MuRP	$46d + 42$
AttH	$59d + 42$
ConE	$188d + 113$
DistMult	$3d$
ComplEx	$12d + 3$
ComplEx-N3	$12d + 3$
TuckER	$2(d_e^2 d_r + d_e d_r + d_e)$
QuatE	$48d + 3$
DualE	$128d$
R-GCN	$d(2bL(B + 1) + 2L + 3)$
ConvE	$2cd(2s_k + 2d + 4S + 7) + d(12S + 22)$
ConvKB	$7dc$
CompGCN	$d_h(2d(B + 2) + 4S + 7) + d(d + 1) + \kappa_{\text{ConvE}}$
NBFNet	$L(Bd(2d + 7) + d(26d + 4S + 19) + 4) + d_h(4d + 3) + 2$

implementations would require less FLOPs. Neural models have ReLU operators, which have a cost of 1 per dimension, given that $\text{ReLU}(x) = \max(0, x)$. To finish, it is worth noting that variable B, in Table 4, is not consistent across GNNs: for R-GCN and CompGCN, it represents the average number of neighbors, regardless of link direction whereas for NBFNet, only neighbors with *incoming* links are counted.

It is easy to see in the table that dimensionality alone is not a good proxy for the computational cost of KGE models. If one fixes d across models as is done in other large-scale studies [35, 2], there still is a high disparity in terms of FLOPs. In particular, the FLOP count of TuckER and all neural models grows quadratically with d.

4.2 Cost of Inference

Following the same procedure as for model size, we assign hyperparameter values to variables of Table 4 for all 49 experiments found in the literature, in order to compare performance and FLOP count estimates. Our overall objective is to compare models on the entire training procedure, which does not only depend on the computational cost of scoring but also on the size of the input KG and a number of epochs. However, we first study the cost of scoring alone. As mentioned in introduction, some models may be relatively cheap to train but expensive to use at inference time, i.e. when scoring arbitrary triples. Once an AI model is deployed in real-world systems, its inference cost is likely to outweigh training cost, even if the model is retrained on a regular basis [48]. This observation motivates a separate evaluation of KGE models on the cost of scoring.

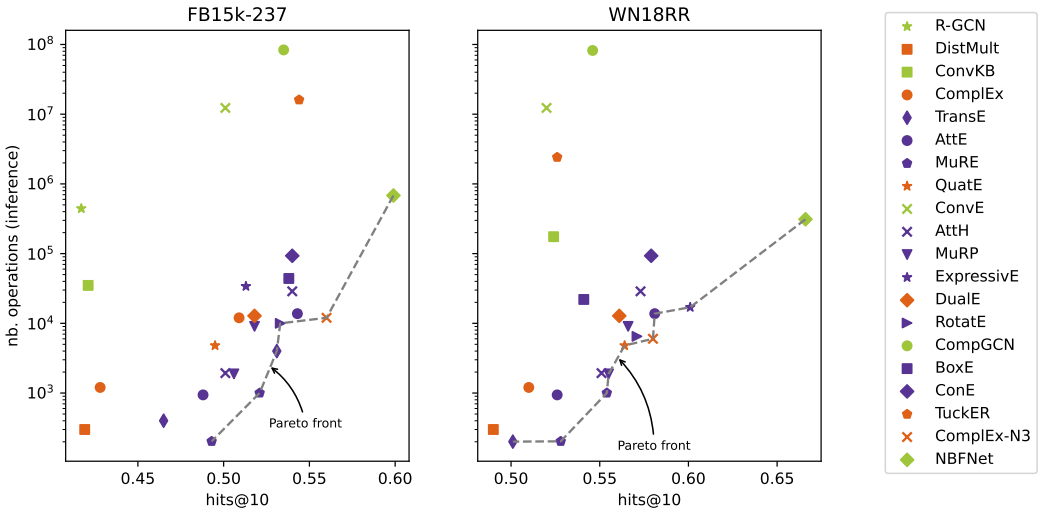


Figure 2 Computational cost of inference with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

FLOP counts for scoring are shown in Figure 2, for FB15k-237 and WN18RR. These results allow to draw conclusions on the relative cost of neural models, on the efficiency of simple models (TransE, ComplEx) and on the cost of hyperbolic models. We comment each aspect in turn.

The figure highlights again the heterogeneity of KGE models, as does Figure 1 on parameter counts. However, we see significant differences across model families. All neural models are in the upper half of the figure (above 50k FLOPs), for both datasets and conversely, all linear or bilinear models but one are in the lower half. The high inference cost of neural models seems not to bring particular benefits: besides NBFNet (which outperforms all other models), neural models have competitors that are both more accurate and more efficient. The same applies to TuckER, the only non-neural model having an inference cost above 50k FLOPs. The main difference between neural models and TuckER is that the high cost of TuckER is highly correlated with its number of parameters, having a spatial complexity in $O(d_e^2 d_r)$, whereas neural models generate many FLOPs from few parameters. The computational overhead of GNNs in particular may be the price to pay for higher inductive abilities, as proven empirically on the inductive link prediction task [52]. The case of NBFNet is also interesting as it improves performance at the expense of a significantly higher inference cost compared to ComplEx-N3 and ExpressivE, the second best performing models on FB15k-237 and WN18RR. We come back to this observation in the next section about training cost.

TransE and MuRE, the simplest linear models, appear to be near-optimal: they reach a slightly lower performance than other linear models but with a much lower inference cost. Interestingly, TransE remains efficient in high dimensions, reaching a hits@10 of 0.53 for a cost of 4k FLOPs on FB15k-237[9]. MuRE appears even closer to optimality, although available data is too sparse to come to conclusions. A similar comment can be made with the bilinear ComplEx model, which also remains efficient in high dimensions, though to a lesser extent. What is worth noting about ComplEx is that its N3 variant, with the same number of parameters and the same inference cost, is able to outperform ComplEx by a large margin (again on FB15k-237). In the next section, we show that it is at the expense of a higher training cost.

[9] The lower-dimensional TransE model for FB15k-237 was not trained with self-adversarial sampling, which could have further improved performance.

Once we add the computation criterion to link prediction, hyperbolic models do not seem to stand out. ConE has a clearly higher inference cost than any other linear model, despite good results. Other hyperbolic models are also an order of magniture more costly than their Euclidian couterpart. The inference cost of MuRP, in particular, is 9 times higher than the inference cost of MuRE. What the figure also shows is the domination of MuRE over hyperbolic models: the high-dimensional MuRE (200 dimensions) beats the lower-dimensional AttH and MuRP (32 dimensions) both in performance and efficiency. Balažević *et al.* and Chami *et al.* independently give evidence that beyond 100 dimensions, the superiority of hyperbolic operators vanishes [8, 13]. That is, for a given computational budget, it may generally hold that Euclidian models are more accurate than hyperbolic models. We further evaluate KGE models in fixed-budget experiments in Section 5, to answer this question.

4.3 Cost of Training

The observations we have made so far are on the computational cost of inference. They may not necessarily hold for training. For instance, a high-dimensional Euclidian model may be harder to train than low-dimensional hyperbolic models. That is why we now look at training cost, by considering a further variable in Formula 10: the number of examples in the training dataset. Across experiments, this value varies greatly because of the N_τ hyperparameter, in the definition of $\mathcal{L}(\tau)$ (Formula 8). Early experiments set $|N_\tau| = 1$, others set it to 10, 50 or even 1,024 for RotatE [40]. Moreover, some models are trained with a slightly different loss formulation that takes all entities into account in a single pass, starting with Dettmers *et al.* [18, 22, 9, 45]. Instead of calculating the loss term from $|N_\tau| + 1$ triples ($|N_\tau|$ negative triples for one positive triple), the alternative is to calculate it from $|E|$ triples and consider the link prediction problem as multi-class classification. We approximate this configuration by setting $|N_\tau| = |E| - B$. Knowing that $|E| = 14,541$ for FB15k-237 and $|E| = 40,943$ for WN18RR, the way negative triples are generated has a significant impact on the total number of FLOPs for training.

To have a cost for the entire training procedure, the number of epochs must also be taken into account (Formula 10, p. 12). In this paper, we consider the cost of a single epoch. Most papers do not report an exact number of epochs for their experiments. Some papers report a maximum number of epochs but it is common to train with early stopping: every x epochs, the model is evaluated on a validation set and if some metric reaches a predefined threshold, training stops immediately. The actual number of training epochs could be much lower than the reported maximum. ComplEx-N3, for instance, was "trained for 100 epochs to ensure convergence" but "reported performances were reached within the first 25 epochs" [22]. In the absence of any more precise data, we calculate the cost of computing $\mathcal{L}(\tau)$ for a *single* triple τ. When we compare KGE models on such a cost, we implicitly assume that stochastic gradient descent has the same convergence rate across models, which is often not true in practice. However, if the single-triple training cost of two models differs by several orders of magnitude, we can safely consider that their true training cost also differs significantly.

Moreover, our estimates ignore entirely any fine-tuning that has taken place to ensure optimality. In the domain of language models for instance, systematic architecture search on a Transformer model required more than 3,000 repeated training phases to improve results [39]. We exclude these considerations here.

The cost of computing $\mathcal{L}(\tau)$, as defined on p. 7 (Formula 8), is $(|N_\tau| + 1)(\kappa_{f(\tau)} + 5)$ in the case $w_{\tau'}$ is a constant, where $\kappa_{f(\tau)}$ is the cost of scoring a single triple. Computing softmax in the self-adversarial setting costs $3|N_\tau|$ FLOPs. We can see in Figure 2 that $\kappa_{f(\tau)} \gg 3$ (TransE and MuRE have the lowest inference cost at 200 FLOPs), which makes the computation of softmax rather cheap compared to standard cross-entropy loss. We choose to ignore softmax in

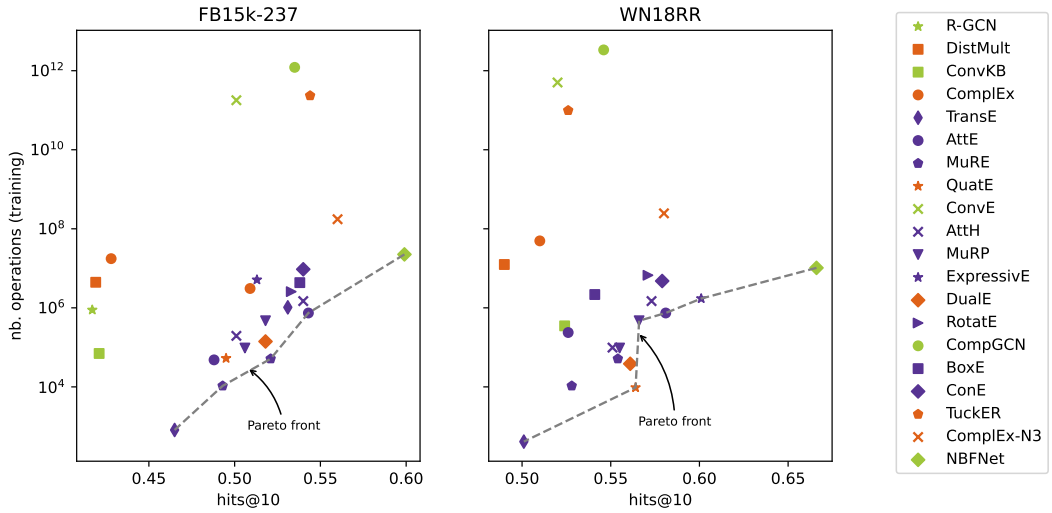


Figure 3 Computational cost of training (per triple) with respect to hits@10 score (purple: linear models, orange: bilinear models, green: neural models).

our calculation. The L2 and N3 regularizers of ComplEx, introduced on p. 8 (Formula 9), can be ignored with a similar argument. Their cost is respectively of $12d$ and $15d$ FLOPs, of the same order of magnitude as $\kappa_{\text{ComplEx}} = 12d + 3$, but ComplEx has only been trained in configurations generating a high number of negative triples. These two observations allow us to reduce the calculation of training cost to $\kappa_{\mathcal{L}(\tau)}$, with marginal estimation error.

Once we define the per-experiment hyperparameter $|N_\tau|$, we have an estimate of the total number of FLOPs for training, which we can then put in perspective with the performance of a model. Results are given in Figure 3, which we can compare to the previous figure on inference cost. As an example, it seems that R-GCN and NBFNet can be efficiently trained despite high inference cost. R-GCN is at the level of AttE (700-800k FLOPs) and NBFNet requires no more than twice as many FLOPs as ConE, the closest linear model. However, no experiment in the literature indicates whether NBFNet can beat linear models at a given training cost. The cost reported here is already for low 32-dimensional embeddings, with few negative triples ($|N_\tau| = 32$). Overall, the training cost of GNN models (R-GCN, NBFNet and CompGCN, in our study) seem not to follow any clear pattern. R-GCN and NBFNet have respectively the lowest and highest hits@10 for a moderate increase in computational cost ($\times 22$ increase for FB15k-237, to compare with the $\times 1,000$ increase between the two TransE experiments on this dataset) while CompGCN has the highest training cost but a rather low performance. GNNs deserve further evaluations on the link prediction task.

The general behavior of bilinear models is also unclear. DualE, for instance, seems relatively efficient when looking at training cost: it requires more FLOPs for scoring than most linear models but its overall training procedure was more effective than those same linear models. QuatE, on WN18RR, is already more efficient at inference time but the gap with linear models similarly increases for training. In contrast to these two models, most experiments on ComplEx and ComplEx-N3 give high training costs for a relatively low inference cost. One exception, published by Sun *et al.* [40], gives good results for ComplEx with 3M FLOPs for training, comparable to the cost of RotatE and ExpressivE. At first sight, it might suggest that other ComplEx and ComplEx-N3 experiments sampled an unnecessarily large number of negative triples. However, comparing the two models at equal dimensions gives a more contrasted picture. On FB15k-237,

ComplEx and ComplEx-N3 were both evaluated with 1,000 dimensions. Given our formalization and simplifications, the only difference between their cost estimates is the number of negative triples they use. For ComplEx, $|N_\tau| = 256$ while for ComplEx-N3, which uses the multi-class classification setting, $|N_\tau| = 14540$. If performance gains come solely from more negative triples, these are not superfluous and ComplEx necessarily comes with a high computational cost. If instead, performance gains come from N3 regularization, further experiments with self-adversarial sampling should be performed to quantify the optimal cost of ComplEx-N3.

5 Fixed-Budget Performance

Evaluations of KGE models found in the literature leave many questions open, including: can the training cost of NBFNet be reduced, is MuRE on the Pareto front of performance/cost criteria, are hyperbolic model intrinsically more expressive than higher-dimensional Euclidian model and what is the true impact of N3 regularization? Some models have already been compared with equal dimensions or equal parameter count but to answer this type of questions, they should be compared with equal *computational budget*. We now provide experiments where model hyperparameters are being set in such a way that all models have the same training cost (as per our approach).

5.1 Approach

According to Figure 3, only two models have a cost below 10k FLOPs (TransE, QuatE). We take this value as the maximum computational budget in our experiments, which gives an upper bound to d, $|N_\tau|$ and other variables of Table 4. Let us first set $|N_\tau| = 1$ and take the example of TransE. We have $\kappa_{\mathcal{L}(\tau)} = (|N_\tau| + 1)(\kappa_{f_{\text{TransE}}(\tau)} + 5) = 2((4d + 1) + 5) = 8d + 12$. Setting $\kappa_{\mathcal{L}(\tau)} = 10,000$, the maximum number of dimensions for TransE is $d = \lfloor (10,000 - 12)/8 \rfloor = 1248$. If instead, we take ComplEx, we have $\kappa_{\mathcal{L}(\tau)} = 2((12d + 3) + 5) = 24d + 16$ and $d = \lfloor (10,000 - 16)/24 \rfloor = 416$. Of course, instead of limiting the number of negative triples, we can also set an arbitrary number of dimensions, e.g. $d = 100$, and adjust $|N_\tau|$ so that a smaller model is trained on more data. For a 100-dimensional TransE, the maximum number of negative triples would be $|N_\tau| = \lfloor (10,000/406) - 1 \rfloor = 23$. For ComplEx, it would be $|N_\tau| = \lfloor (10,000/1208) - 1 \rfloor = 7$. We consider both configurations in our experiments, as they play different roles. When setting the same number of negative triples across models, we can compare models with each other and assess their efficiency – up to inherent limitations in expressivity of each model. When setting a fixed, arbitrary number of dimensions, we can compare a model with itself, showing how the dimensionality of the KG (through negative sampling) also influences training.

The cost estimate of neural models involves other free variables: b, d_h, c, s_k, L and S. We aim to assign them to the lowest possible values, such as $b = 2$, $L = 2$, $c = 2$ or $c = 4$, $S = 128$. It happens that CompGCN, NBFNet and ConvE have no satisfactory variable assignment that keeps the cost of training under 10k FLOPs. CompGCN and NBFNet have hidden layers with dimensions d_h but in practice, all experiments set $d_h = 2d$ for these models. The convolutional filter of ConvE must have a minimal size $s_k = 3 \times 3$. With these two values, even with a single negative triple, scoring is too costly. The only neural models that can be evaluated with a budget of 10k FLOPs are R-GCN and ConvKB. ConvKB embeddings depend on a number of convolution filters, c. If $c = 1$, the formalization of the model is analogous to a TransE-like linear model, which encourages to find instead a value above 1. We set $c = 4$, which is the largest value such that $d > 100$ in the configuration where $|N_\tau| > 1$. In the original experiments, ConvKB models have 50 to 500 filters [29]. R-GCN experiments, on the other hand, have set $b = 5$ and $L = 2$. With such a configuration, R-GCN would also be over the 10k computational budget. We decided to reduce c and let $L \in \{1, 2\}$. In fact, already if $c = 1$, R-GCN must have low-dimensional embeddings to remain under 10k FLOPs ($d < 32$). We therefore take $c = 1$ in all R-GCN experiments.

For each model, we have at least two configurations: one has $|N_\tau| = 1$, the others have $|N_\tau| > 1$. R-GCN has 4 configurations per dataset, as we also evaluate the influence of L. MuRP and AttH have 3 configurations, to better evaluate the impact of negative triples on hyperbolic models. All configurations are summarized in Table 5. For configurations with multiple negative triples, d has a maximum value of 100. If there can be no configuration, for a given model, such that $d = 100$ and $|N_\tau| > 1$, we set $|N_\tau| = 2$ or $|N_\tau| = 4$ and find the highest possible value for $d \in [32, 100]$. If no assignment for d exists in this interval, we discard the model (which applies to ConE, TuckER and DualE).

In total, we obtain 64 distinct fixed-budget experiments. We set $S = 128$, a common value across experiments from the literature (43% of experiments listed in Tables 6 and 7). Linear models were trained with $\lambda = 3$, also common for small dimensions; other models with $\lambda = 0$. Any other hyperparameter was set manually. In particular, we set a learning rate of 10^{-3} and run training for 50 epochs. Note that despite our effort to find suitable values for model-specific hyperparameters, we do not strictly guarantee that the reported performance is the best possible performance with a 10k computational budget.

For our evaluation, we used different KGE libraries, all based on PyTorch. The PyKEEN library[10], the most up-to-date KGE library [3], implements 11 of the 20 models we review in this paper (8 included in this section's experiments). To evaluate AttE, AttH and ComplEx-N3, we used the KGEmb library[11], which specifically targets hyperbolic KGE models [13]. For the remaining models, ExpressivE and MuRP, we used original implementations provided respectively by Pavlović[12] and Balažević[13]. We have made configuration scripts available to install these libraries and reproduce our fixed-budget experiments[14].

5.2 Model Performance

Table 5 gives the performance of KGE models along two metrics: hits@10 and Mean Rank (MR, lower is better). On both metrics, results show a clear advantage for linear models. With few exceptions, all linear models outperform any of the bilinear or neural models, by a large margin. Among linear models, the only exceptions are TransE, which performs poorly with a single negative triple, and RotatE, which is beaten by R-GCN on a single metric (hits@10 on FB15k-237). Conversely, QuatE performs relatively well on WN18RR, even if its MR is worse than the average MR among linear models. ComplEx-N3 and DistMult are the worst performing models, regardless of the configuration. Our results suggest that the N3 regularization scheme imposes a high computational cost, contrary to the classical L2 regularization used on ComplEx. It might however also be that better-suited hyperparameters yield good results for ComplEx-N3.

If we focus on linear models only, we can make two observations. First, despite a certain diversity in their parameterization, linear models have relatively close results. Hits@10 has a standard deviation below 0.05 on both datasets, if we exclude the first TransE configuration. Notably, MuRE and AttE have very close results despite 6 times more parameters for MuRE (with $|N_\tau| = 1$). Second, hyperbolic models (MuRP, AttH) are consistently outperformed by their Euclidian counterpart, even though the margin reduces as the training set includes more negative triples. In other words, simple linear models, such as MuRE or RotatE (on WN18RR only), appear to be able to compensate the richer geometric operations of other models simply with more dimensions – up to a certain number of dimensions.

[10] https://github.com/pykeen/pykeen
[11] https://github.com/HazyResearch/KGEmb/
[12] https://github.com/AleksVap/ExpressivE
[13] https://github.com/ibalazevic/multirelational-poincare
[14] https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments

Chami *et al.* report that MuRP and AttH achieve 0.544 and 0.551 hits@10 on WN18RR in their 32-dimensional version (ConE achieves 0.537 hits@10 under the same conditions). These results are well above those we find with more dimensions but they hold for $|N_\tau| = 50$. The computational cost of training the 2 models in this setting is 77,469 and 98,685 FLOPs – well beyond the 10k budget. We may find comparable results if we further reduce d to increase $|N_\tau|$, though. Going from 1 negative triple to 2 and 4 negative triples does indeed improve the performance of MuRP and AttH on FB15k-237 but not on WN18RR. Only BoxE follows the same pattern. ConvKB has a lower performance on both datasets; R-GCN has contrasted results if we compare $|N_\tau| = 1$ and $|N_\tau| = 2$. All other models improve if they are trained with more negative triples. MuRE, for instance, improves its hits@10 by 5.8% with more than 10 negative triples. This configuration is in fact the best overall: it is only beaten by RotatE with respect to hits@10 on WN18RR. A recent re-evaluation of MuRE, although with a different objective, arrives at a similar conclusion: with equal training conditions, MuRE are RotatE are among the best-performing models [15].

In the light of fixed-budget experiments, we can identify strengths and weaknesses of each of the KGE model families:

- Euclidian models are the most efficient models: they achieve good results with a low computational budget. MuRE appears to be the most efficient model overall. It deserves further experiments in high dimensions to better understand its potential limitations in terms of expressivity.
- Hyperbolic models display no performance gain for a fixed low budget. As the literature points that the hyperbolic operators offer no advantages in high dimensions, these models are likely to have a limited impact in downstream tasks. In contrast though, there is no theoretical work on how high-dimensional Euclidian models can efficiently capture hierarchies (as in WN18RR). Our results encourage further work in this area.
- Bilinear models require more computation than linear models, with no obvious gain in expressivity. This result questions the relevance of using ComplEx in other KGE learning tasks, such as query answering [5]; high-dimensional linear models might prove better suited. The performance of QuatE on FB15k-237 could still be further studied, in case hyperparameter tuning alone would improve its performance. The same applies to N3 regularization.
- Non-linearities, as introduced in ConvKB, R-GCN and other neural models, appear to be ineffective for low computational budgets. It is however still unclear whether they are necessary to increase the expressivity of KGEs. Interestingly, to match the (high) inference cost of NBFNet, MuRE could have more than 50,000 dimensions, i.e. more than the number of entities in the input KG. With such a large embedding space, it would most likely be more efficient to consider non-parametric approaches instead, along the same line as AnyBURL.

6 Conclusion

The main motivation of this paper was to quantify the computational cost of the numerous KGE models already published and to identify research directions that would take into account the *efficiency* of learning on KGs. To this end, we designed a simple cost model that estimates the number of FLOPs each KGE model would require for scoring and training, and compared the resulting estimates over 49 experiments. The first conclusion to draw from this analysis is that the efficiency of KGE models greatly varies across experiments. Most bilinear and neural models meet the performance of linear models only at the expense of a higher computational cost. In particular, state-of-the-art results for ComplEx and ConvE have only been given in a setting exposing much more data to the model than linear models – because of negative sampling. Fixed-budget experiments suggest that, when given an equal number of negative triples, bilinear

and neural models do not compete with linear models. Future research should either focus on expanding linear models or finding an equivalent formulation of ComplEx and QuatE scoring as a linear fonction.

On the other hand, linear models (TransE, RotatE, BoxE, etc.) have known limitations with respect to their ability to embed ontological knowledge and rules, regardless of their dimensionality. The exceptional performance of NBFNet suggest that GNN architectures may have richer inductive abilities, despite higher computational costs. It is however a still open question whether message passing and non-linearities are necessary. For instance, MuRE, which stands out as both accurate and efficient for transductive link prediction, has not been tested on other KG learning tasks such as inductive link prediction, complex query answering or ontology embedding. Interestingly, MuRE itself was not the main subject of its original publication (MuRP was) but fixed-budget experiments highlight its superior efficiency for low computational budgets. A higher-dimensional MuRE model (beyond 200 dimensions) might as well surpass GNNs if properly trained.

Analyzing the computational cost of KGE models, as we do in this paper, is an important research objective as we anticipate that large KGs will be commonly used through pre-trained embeddings. Training embeddings for the 100M+ entities of Wikidata, for example, would have a significant cost; minimizing this cost then becomes crucial. It is worth noting, still, that our cost model and experiments have several limitations. First, FLOP estimates are only loosely correlated with execution time and energy consumption. Typically, this means that ComplEx and MuRE can be trained on similar computing platforms, despite significant differences in FLOPs. If the two models were compared on a more elaborate cost model, also taking spatial complexity into account, results might be more contrasted than what we expose in the paper. In addition, the fixed-budget experiments we conducted only explore sparse configurations. A more thorough study to find scaling laws, relating dataset size and cost or cost and hits@10, would provide more insights as to what models are most efficient. We aim to cover such scaling experiments in future work.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A Box Embedding Model for Knowledge Base Completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/6dbbe6abe5f14af882ff977fc3f35501-Abstract.html.

2 Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing Light Into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(12):8825–8845, 2022. doi:10.1109/TPAMI.2021.3124805.

3 Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021. URL: http://jmlr.org/papers/v22/20-825.html.

4 Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute, 2018. URL: https://openai.com/research/ai-and-compute.

5 Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations*, 2021. URL: https://openreview.net/forum?id=Mos9F9kDwkz.

6 Yushi Bai, Zhitao Ying, Hongyu Ren, and Jure Leskovec. Modeling Heterogeneous Hierarchies with Relation-specific Hyperbolic Cones. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 12316–12327, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/662a2e96162905620397b19c9d249781-Abstract.html.

7 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. Hypernetwork knowledge graph em-

beddings. In Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2019 - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings - Workshop and Special Sessions*, volume 11731 of *Lecture Notes in Computer Science*, pages 553–565. Springer, 2019. doi:10.1007/978-3-030-30493-5_52.

8 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. Multi-relational Poincaré Graph Embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4465–4475, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/f8b932c70d0b2e6bf071729a4fa68dfc-Abstract.html.

9 Ivana Balažević, Carl Allen, and Timothy M. Hospedales. TuckER: Tensor Factorization for Knowledge Graph Completion. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 5184–5193. Association for Computational Linguistics, 2019. doi:10.18653/v1/D19-1522.

10 Zhen Bi, Siyuan Cheng, Jing Chen, Xiaozhuan Liang, Feiyu Xiong, and Ningyu Zhang. Relphormer: Relational graph transformer for knowledge graph representations. *Neurocomputing*, 566:127044, 2024. doi:10.1016/J.NEUCOM.2023.127044.

11 Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013. URL: https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

12 Zongsheng Cao, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Dual Quaternion Knowledge Graph Embeddings. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6894–6902. AAAI Press, 2021. doi:10.1609/AAAI.V35I8.16850.

13 Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-Dimensional Hyperbolic Knowledge Graph Embeddings. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6901–6914. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.617.

14 Victor Charpenay. KGE Cost Experiments. Software, swhId: swh:1:dir:29d727e535031eca0583c489c67c47d0b9165d30 (visited on 2026-03-27), URL: https://gitlab.emse.fr/victor.charpenay1/kge-cost-experiments/. doi:10.4230/artifacts.25692.

15 Victor Charpenay and Steven Schockaert. Less is MuRE: Revisiting shallow knowledge graph embeddings. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2025. doi:10.18653/V1/2025.EMNLP-MAIN.779.

16 Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. HittER: Hierarchical transformers for knowledge graph embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10395–10407, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.emnlp-main.812.

17 Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL: http://jmlr.org/papers/v25/23-0870.html.

18 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018. doi:10.1609/AAAI.V32I1.11573.

19 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019,*

Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/V1/N19-1423.

20 Jiabang He, Liu Jia, Lei Wang, Xiyao Li, and Xing Xu. MoCoSA: Momentum Contrast for Knowledge Graph Completion with Structure-Augmented Pre-trained Language Models. *CoRR*, abs/2308.08204, 2023. arXiv: 2308.08204. doi:10.48550/arXiv.2308.08204.

21 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Springer, 2021. doi:10.2200/S01125ED1V01Y202109DSK022.

22 Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical Tensor Decomposition for Knowledge Base Completion. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 2018. URL: http://proceedings.mlr.press/v80/lacroix18a.html.

23 Da Li, Ming Yi, and Yukai He. LP-BERT: Multi-task Pre-training Knowledge Graph BERT for Link Prediction. *CoRR*, abs/2201.04843, 2022. arXiv: 2201.04843. arXiv:2201.04843.

24 Haotian Li, Bin Yu, Yuliang Wei, Kai Wang, Richard Yi Da Xu, and Bailing Wang. Kermit: Knowledge graph completion of enhanced relation modeling with inverse transformation. *Knowledge-Based Systems*, 324:113500, 2025. doi:10.1016/j.knosys.2025.113500.

25 Yunshui Li, Junhao Liu, Min Yang, and Chengming Li. Self-distillation with meta learning for knowledge graph completion. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 2048–2054. Association for Computational Linguistics, 2022. doi:10.18653/V1/2022.FINDINGS-EMNLP.149.

26 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, 2015. doi:10.1609/AAAI.V29I1.9491.

27 Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of BLOOM, a 176B parameter language model. *Journal of Machine Learning Research*, 24(253):1–15, 2023. URL: http://jmlr.org/papers/v24/23-0069.html.

28 Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3137–3143. ijcai.org, 2019. doi:10.24963/ijcai.2019/435.

29 Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Q. Phung. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 327–333. Association for Computational Linguistics, 2018. doi:10.18653/v1/n18-2053.

30 Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016. doi:10.1109/JPROC.2015.2483592.

31 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress, 2011. URL: https://icml.cc/2011/papers/438_icmlpaper.pdf.

32 Simon Ott, Christian Meilicke, and Matthias Samwald. SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models. In Danqi Chen, Jonathan Berant, Andrew McCallum, and Sameer Singh, editors, *3rd Conference on Automated Knowledge Base Construction, AKBC 2021, Virtual, October 4-8, 2021*, 2021. doi:10.24432/C5MK57.

33 Aleksandar Pavlovic and Emanuel Sallinger. ExpressivE: A Spatio-Functional Embedding For Knowledge Graph Completion. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=xkev3_np08z.

34 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Trans. Knowl. Discov. Data*, 15(2):14:1–14:49, 2021. doi:10.1145/3424672.

35 Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BkxSmlBFvr.

36 Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang,

Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. BLOOM: A 176B-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022. doi:10.48550/arXiv.2211.05100.

37 Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018. doi:10.1007/978-3-319-93417-4_38.

38 Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020. doi:10.1145/3381831.

39 Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650. Association for Computational Linguistics, 2019. doi:10.18653/v1/p19-1355.

40 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HkgEQnRqYQ.

41 Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha P. Talukdar, and Yiming Yang. A Re-evaluation of Knowledge Graph Completion Methods. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5516–5522. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.489.

42 Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Alexandre Allauzen, Edward Grefenstette, Karl Moritz Hermann, Hugo Larochelle, and Scott Wen-tau Yih, editors, *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, CVSC 2015, Beijing, China, July 26-31, 2015*, pages 57–66. Association for Computational Linguistics, 2015. doi:10.18653/v1/W15-4007.

43 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi:10.48550/arXiv.2302.13971.

44 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/trouillon16.html.

45 Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BylA_C4tPr.

46 Feiyang Wang, Zhongbao Zhang, Li Sun, Junda Ye, and Yang Yan. DiriE: Knowledge graph embedding with Dirichlet distribution. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 3082–3091. ACM, 2022. doi:10.1145/3485447.3512028.

47 Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Trans. Knowl. Data Eng.*, 29(12):2724–2743, 2017. doi:10.1109/TKDE.2017.2754499.

48 Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim M. Hazelwood. Sustainable AI: environmental implications, challenges and opportunities. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu, editors, *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022. URL: https://proceedings.mlsys.org/paper_files/paper/2022/hash/462211f67c7d858f663355eff93b745e-Abstract.html.

49 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May*

7-9, 2015, Conference Track Proceedings, 2015. URL: http://arxiv.org/abs/1412.6575.

50 Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion Knowledge Graph Embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2731–2741, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/d961e9f236177d65d21100592edb0769-Abstract.html.

51 Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3065–3072. AAAI Press, 2020. doi:10.1609/AAAI.V34I03.5701.

52 Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29476–29490. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/f6a673f09493afcd8b129a0bcf1cd5bc-Paper.pdf.

A Cost of Pre-Trained Language Models

We use the calculation method first elaborated by OpenAI researchers and then applied on the Transformer architecture by EpochAI researchers[15]. A Transformer layer is mainly composed of two elements: a multi-head attention network (MHA) and a feed-forward network (FFN). The number of FLOPs of MHA, for a single token, is given by EpochAI as:

$$\kappa_{MHA} = 2H(d_i(2d_k + d_h) + (d_k + d_h) + d_h d_o) \tag{12}$$

where H is the number of heads and d_i, d_k, d_h, d_o are respectively the dimension of the input token embedding, of the key embedding, of the hidden vector and of the output vector of the MHA unit.

The number of FLOPs of a FFN with two layers, as is typical in Transformers, is:

$$\kappa_{FFN} = 2d_o d'_h + 2d'_h d_o \tag{13}$$

where d'_h is the dimension of the network's hidden vector.

The hyperparameters given by Devlin *et al.* for BERT base are the following [19]: $H = 12, d_o = 768, d'_h = 4 \times 768 = 3,072$. Values for other dimensions are not explicitly given but we assume $d_i = d_k = d_h = 64$, as in the original Transformer. As a result, we have $\kappa_{MHA} = 1.5 \times 10^6$ and $\kappa_{MHA} = 9.4 \times 10^6$ for a single BERT layer. BERT base having 12 layers, we have $\kappa_{BERT} = 12 \times (1.5 + 9.4) \times 10^6 = 1.3 \times 10^8$.

B Experiment Hyperparameters

We list all experiments with known hits@10 and hyperparameters in Tables 6 (for FB15k-237) and 7 (for WN18RR). We used these values to produce Figures 1, 2 and 3.

C Execution Time of Atomic Operations

We use execution time as a proxy for the number of FLOPs of atomic operations to evaluate the practical relevance of our estimates (Section 4.2). We report values for computation on a CPU and on a GPU, respectively on Figure 4 and Figure 5. Times have been measured for 100 iterations over 1,000-dimensional random vectors in both cases.

[15] https://epoch.ai/blog/estimating-training-compute#example-transformer

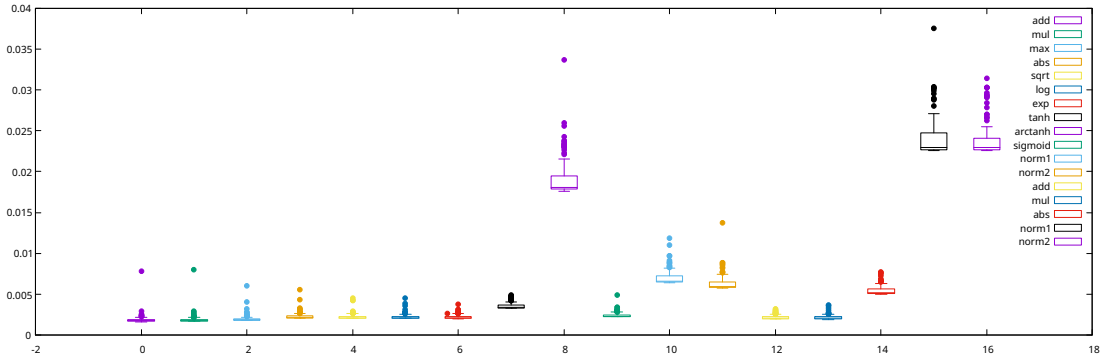


Figure 4 Execution time of atomic operations on random PyTorch tensors, on a CPU.

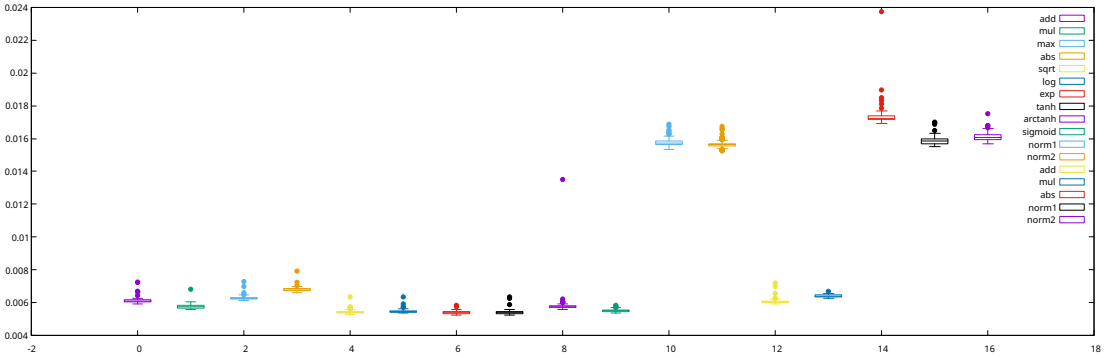


Figure 5 Execution time of atomic operations on random PyTorch tensors, on a GPU.

Table 5 Model performance for a 10k computational budget (d: dimension of embeddings, $|N_\tau|$: nb. of negative triples per KG triple, L: nb. of GNN layers).

				FB15k-237		WN18RR	
Model	**d**	**$\|N_\tau\|$**	**L**	**MR**	**hits@10**	**MR**	**hits@10**
TransE	1248	1	–	2402	0.173	11379	0.044
	100	23	–	247	0.371	1289	0.477
RotatE	499	1	–	301	0.311	1566	0.564
	100	8	–	219	0.409	3244	0.577
BoxE	113	1	–	219	0.349	2905	0.545
	75	2	–	199	0.384	3164	0.530
ExpressivE	146	1	–	272	0.347	3653	0.469
	97	2	–	208	0.381	4935	0.472
MuRE	998	1	–	189	0.409	2059	0.539
	100	18	–	216	0.467	2768	0.562
MuRP	107	1	–	209	0.400	2377	0.477
	71	2	–	190	0.429	2235	0.495
	42	4	–	185	0.441	2350	0.489
AttE	170	1	–	204	0.417	3061	0.523
	100	2	–	190	0.435	3132	0.509
AttH	82	1	–	212	0.413	3211	0.516
	54	2	–	199	0.430	3368	0.505
	33	4	–	198	0.436	3693	0.469
DistMult	1665	1	–	1171	0.188	15825	0.073
	100	31	–	1108	0.179	18251	0.029
ComplEx	416	1	–	1358	0.181	11359	0.323
	100	7	–	838	0.252	7631	0.432
ComplEx-N3	416	1	–	6979	0.001	13828	0.003
	100	7	–	3305	0.042	14541	0.013
QuatE	104	1	–	723	0.267	5290	0.524
	69	2	–	640	0.272	5526	0.527
R-GCN	53	1	1	307	0.318	–	–
	27	1	2	306	0.290	–	–
	35	2	1	276	0.321	–	–
	18	2	2	309	0.293	–	–
	293	1	1	–	–	5954	0.434
	161	1	2	–	–	5511	0.377
	100	4	1	–	–	4851	0.429
	100	2	2	–	–	5848	0.378
ConvKB	178	1	–	397	0.308	11226	0.135
	100	2	–	472	0.251	10635	0.112

Table 6 Hyperparameters found in the literature for experiments on FB15k-237 (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, $|N_\tau|$: nb. of negative triples per KG triple, S: batch size) with indication whether inverse relations are explicitly modeled.

	Model	hits@10	d	$\|N_\tau\|$	S	Inverse	Other HPs
FB15k-237	R-GCN [37]	0.417	500	1	30000	?	$b = 5, L = 2, B = 43$
	DistMult [18]	0.419	100	14540	128	0	
	ConvKB [41]	0.421	100	1	256	0	$c = 50$
	ComplEx [18]	0.428	100	14540	128	0	
	TransE [29]	0.465	100	1	256	0	
	AttE [13]	0.488	32	50	500	1	
	MuRE [8]	0.493	40	50	128	1	
	QuatE [50]	0.495	100	10	27211	0	
	ConvE [18]	0.501	200	14540	128	0	$c = 32, s_k = 3 \times 3$
	AttH [13]	0.501	32	100	500	1	
	MuRP [8]	0.506	40	50	128	1	
	ComplEx [40]	0.509	1000	256	1024	0	
	ExpressivE [33]	0.513	1000	150	1024	0	
	DualE [12]	0.518	100	10	27211	0	
	MuRP [8]	0.518	200	50	128	1	
	MuRE [8]	0.521	200	50	128	1	
	TransE [40]	0.531	1000	256	1024	0	
	RotatE [40]	0.533	1000	256	1024	0	
	CompGCN [45]	0.535	100	14540	128	1	$d_h = 200, c = 200, s_k = 7 \times 7, B = 43$
	BoxE [1]	0.538	1000	100	1024	0	
	ConE [6]	0.54	500	100	1024	0	
	AttH [13]	0.54	500	50	500	1	
	AttE [13]	0.543	500	50	500	1	
	TuckER [9]	0.544	200	1	128	1	$d_e = d_r = d$
	ComplEx-N3 [22]	0.56	1000	1	1000	1	
	NBFNet [52]	0.599	32	32	256	1	$L = 6, d_h = 64, B = 23$

Table 7 Hyperparameters found in the literature for experiments on WN18RR (d, d_e, d_r: size of (entity/relation) embeddings, b: block size, d_h: size of hidden layer, c: nb. of convolution filters, s_k: size of convolutional filter, L: nb. of layers, B: avg. nb. of neighbors, $|N_\tau|$: nb. of negative triples per KG triple, S: batch size) with indication whether inverse relations are explicitly modeled.

	Model	hits@10	d	$\|N_\tau\|$	S	Inverse	Other HPs
WN18RR	DistMult [18]	0.49	100	40942	128	0	
	TransE [29]	0.501	50	1	256	0	
	ComplEx [18]	0.51	100	40942	128	0	
	ConvE [18]	0.52	200	40942	128	0	$c = 32, s_k = 3 \times 3$
	ConvKB [41]	0.524	50	1	256	0	$c = 500$
	TuckER [9]	0.526	200	1	128	1	$d_e = d, d_r = 30$
	AttE [13]	0.526	32	250	100	1	
	MuRE [8]	0.528	40	50	128	1	
	BoxE [1]	0.541	500	100	512	0	
	CompGCN [45]	0.546	100	40942	128	1	$d_h = 200, c = 200, s_k = 7 \times 7, B = 5$
	AttH [13]	0.551	32	50	500	1	
	MuRE [8]	0.554	200	50	128	1	
	MuRP [8]	0.555	40	50	128	1	
	DualE [12]	0.561	100	2	8683	0	
	QuatE [50]	0.564	100	1	8683	0	
	MuRP [8]	0.566	200	50	128	1	
	RotatE [40]	0.571	500	1024	512	0	
	AttH [13]	0.573	500	50	1000	1	
	ConE [6]	0.579	500	50	1024	?	
	ComplEx-N3 [22]	0.58	500	1	100	1	
	AttE [13]	0.581	500	50	1000	1	
	ExpressivE [33]	0.601	500	100	512	0	
	NBFNet [52]	0.666	32	32	128	1	$L = 6, d_h = 64, B = 3$