A Lightweight Knowledge Graph Embedding Framework for Efficient Inference and Storage

Haoyu Wang[§], Yaqing Wang[§], Defu Lian[†⋄] and Jing Gao[§]
[§]Purdue University, West Lafayette, Indiana, USA [†]University of Science and Technology of China
[⋄]Yangtze River Delta Information Intelligence Innovation Research Institute
[§]{wang5346, wang5075, jinggao}@purdue.edu, [†]liandefu@ustc.edu.cn

ABSTRACT

Knowledge graphs, which consist of entities and their relations, have become a popular way to store structured knowledge. Knowledge graph embedding (KGE), which derives a representation for each entity and relation, has been widely used to capture the semantics of the information in the knowledge graphs, and has demonstrated great success in many downstream applications, such as the extraction of similar entities in response to a query entity. However, existing KGE methods cannot work well on emerging knowledge graphs that are large-scale due to the constraints in storage and inference efficiency. In this paper, we propose a lightweight KGE model, LightKG, which significantly reduces storage as well as running time needed for inference. Instead of storing a continuous vector for every entity, LightKG only needs to store a few codebooks, each of which contains some codewords that correspond to the representatives among the embeddings, and the indices that correspond to the codeword selections for entities. Hence LightKG can achieve highly efficient storage. The efficiency of the downstream querying process can be significantly boosted too with the proposed LightKG model as the relevance score between the query and an entity can be efficiently calculated via a quick look-up in a table that contains the scores between the query and codewords. The storage and inference efficiency of LightKG is achieved by its novel design. LightKG is an end-to-end framework that automatically infers codebooks and codewords and generates an approximated embedding for each entity. A residual module is included in LightKG to induce the diversity among codebooks, and a continuous function is adopted to approximate codeword selection, which is non-differential. In addition, to further improve the performance of KGE, we propose a novel dynamic negative sampling method based on quantization, which can be applied to the proposed LightKG or other KGE methods. We conduct extensive experiments on five public datasets. The experiments show that LightKG is search and memory efficient with high approximate search accuracy. Also, the dynamic negative sampling can dramatically improve model performance with over 19% improvement on average.

CIKM '21, November 1–5, 2021, Virtual Event, Australia.

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482224

CCS CONCEPTS

• **Information systems** → **Entity relationship models.**

KEYWORDS

knowledge graph embedding, quantization

ACM Reference Format:
Haoyu Wang, Yaqing Wang, Defu Lian and Jing Gao. 2021. A Light-weight Knowledge Graph Embedding Framework for Efficient Inference and Storage. In *Proceedings of the 30th ACM Int'l Conf. on Information and Knowledge Management (CIKM'21), November 1–5, 2021, Virtual Event, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3459637.3482224

1 INTRODUCTION

A knowledge graph usually consists of triples in the form of (*head entity, relation, tail entity*), which capture the relations between pairs of entities. Such large-scale KGs can provide a huge volume of knowledge for various downstream applications, such as question answering [16, 48], information retrieval [11, 28], recommendation [39, 47], and drug discovery [33]. To leverage the structured knowledge in KGs for these applications, usually an essential step is to convert the knowledge into semantic embeddings [4, 45]. However, the scale of these gigantic KGs imposes significant challenges in both space and time. With millions or even billions of entities, the needed space to store the embeddings of all the triplets could be huge. In addition, the semantic embeddings of knowledge graph triples are often used for retrieval purpose, and thus deployed in tasks, such as knowledge graph completion [4, 10, 27, 36, 45], knowledge graph query [41], and knowledge graph reasoning [7]. When the scale of the KG is too big, the inference based on the embeddings could suffer from high computation complexity. Motivated by these challenges, we aim to develop a lightweight knowledge graph embedding (KGE) framework which can not only reduce the space used to store the embeddings but also speed up the inference and retrieval based on these embeddings.

Although existing KGE methods have demonstrated to be effective, unfortunately it is not trivial to reduce the storage consumption and improve inference efficiency of these methods. These methods usually learn entity and relation embeddings in a continuous vector space. Suppose entity and relation embeddings are d-dimensional, and there are n_e entities and n_r relations. To store entity and relation representations, the space cost is $4n_e d + 4n_r d$ Bytes if using 32 bit float representation. The storage consumption grows linearly with the number of entities and the number of relations with slop $4d$. The computational cost of inference and retrieval based on these embeddings is also high. As an example, let's consider the task of conducting knowledge graph completion, i.e., the inference of missing facts based on existing triples. Consider a triple with missing

tail entity $(h, r, ?)$ and we need to identify this missing entry. To select K entities which are most likely to form the triple, it is necessary to compute a relevance score among h, r and each of the n_e entities and rank the entities according to the scores. KGE methods calculate the relevance score as the Euclidean distance (translation-based methods, such as TransE [4] and TransH [42]) or inner product (bilinear models, such as DistMult [45] and RESCAL [30]) between the embeddings. Therefore, no matter which KGE method is adopted, the time complexity need to search for K entities equals to $O(n_e d + K \log n_e)$, which could be a bottleneck when n_e is very big. With approximate search and compression approaches such as hashing [13] and product quantization [20], acceleration and compression can be achieved. However, these search indexes are learnt independently, which are not related to the process of learning entity and relation representations. Hence, it is possible that the adoption of these acceleration and compression techniques may amplify approximation error and lead to low approximate search accuracy.

In light of these challenges, we propose a lightweight knowledge graph coding scheme, referred to as LightKG, to output approximate but accurate embeddings. Since usually the number of entities contained in the knowledge graph is much larger than the number of relations, we only consider entity embeddings in this paper. LightKG is based on the quantization of embeddings. It divides the original d-dimensional entity embedding into m d/m-dimensional subspace, in which m is a positive integer and it indicates the compression ratio. In each subspace, suppose there are B codebooks which consist of W codewords (d/m-dimensional latent vectors), in which B is a small positive integer. We take the codeword that is the most similar to the entity embedding within a codebook of a subspace. Then we sum up these selected codewords, which is treated as a slice of the entity embedding. The quantized entity embedding is then composed by the concatenation of these slices. By this quantization, for one entity, it can be represented by the ids of its most similar codewords, which can be compactly encoded by $\frac{1}{8} m B \log W$ Bytes. As shown in Fig. 1, where there are 4 subspace and 4 codewords in each of codebooks, the entity embedding is encoded via 2 bits for each codebook, and 4 8-bits unsigned integers. Comparing with $4d$ Bytes needed by traditional KGE methods, a big saving in storage space by LightKG can be achieved. When the output of LightKG is adopted for the inference of missing entities, we just need to compute relevance scores via table look-up instead of exhaustive search. As shown in Fig. 1, the scores between the query vector and codewords are computed and stored in look-up tables, and then to calculate the final relevance scores, we just need to conduct a quick look up. The complexity could be $O(n_e + BWd)$.

However, it is difficult to train LightKG end-to-end directly since it involves the selection of similar codewords, which is non-differentiable and cannot be optimized via gradient descent. To solve this problem, we use tempered softmax [19, 29] to approximate argmax used in codeword selection, and resort to the straight-through estimator [3] to compute gradients. Another challenge in the training of LightKG is how to prevent codebooks becoming similar. In each subspace, there are multiple codebooks. If there is no constraint, it is likely to get extremely similar or even the same codebooks, which may make the model less expressive. To solve this challenge, we propose a residual module to ensure the diversity of codebooks.

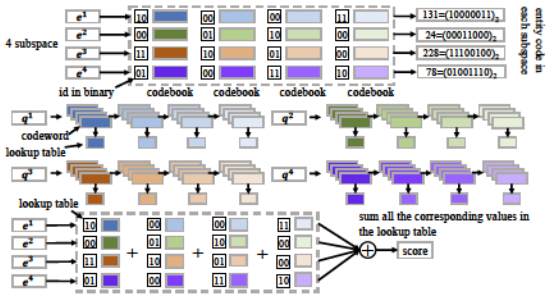


Figure 1: Illustration of Efficient Inference

To further improve the performance of LightKG, we propose a dynamic negative sampling method. The training of KGE methods requires both positive and negative samples in the training set. Positive samples are simply those triples in the knowledge graph, but from the huge space of possible negative samples (those nonexistent triples), how to conduct negative sampling is a very important step. Previously, people usually sample negative samples uniformly [4]. However, it is difficult to obtain informative negative samples in this way. To obtain high-quality negative samples, we design a sampling strategy as follows. We define the probability that a negative sample is high-quality based on its relevance score. Sampling from this distribution increases the chances of getting informative samples, but the pmf (probability mass function) of this distribution is computationally complex [23]. To solve this problem, we propose an approximate distribution, which can be computed very efficiently based on clustering. As negative sampling prepares the training data for KGE methods, the proposed new sampling strategy applies to not only LightKG but also TransE, DistMult and other KGE methods.

The contributions in this paper are summarized as follows: 1) We propose a lightweight KGE framework–LightKG, which achieves storage and inference efficiency with only minor loss in accuracy, 2) We propose a general dynamic negative sampling method, which can efficiently obtain high-quality negative samples for effective training of KGE methods, and 3) We conduct extensive experiments on multiple public datasets to evaluate the proposed LightKG and dynamic negative sampling method. The results reveal that both of them are effective and lead to significant improvements compared with state-of-the-art baselines.

2 RELATED WORK

In this section, we review works related to our task including KGE, lightweight KGE and negative sampling for knowledge graphs.

Usually, KGE methods can be classified as translation-based models [4, 27, 36, 42], bilinear models [30, 38, 45] and deep learning-based models [10, 34]. Translation-based models usually defines the relevance scores according to the Euclidean distance or Manhattan distance applied to embeddings. Bilinear models apply bilinear functions to calculate relevance scores. The score of a triple is defined as: $\boldsymbol{h}^T R t$, , where $\boldsymbol{h}$ is the head entity vector, t is the tail entity vector, and R is the relation matrix. Recently, deep learning-based models are developed. For example, ConvE [10] uses multi-layer convolutional network to learn entity and relation representations, and R-GCN [34] employs graph neural network. In this paper, we

set TransE (one of the most representative translation-based methods) and DistMult (one of the most representative bilinear methods) as the backbone of LightKG, and we also show that the proposed LightKG can be extended to deep learning-based model ConvE.

The development of lightweight models has received increasing attention, and well-known approaches include binarization network [17, 31], product quantization [12, 20], and low precision quantization [9, 18]. Few of them focus on KGE and they cannot be directly applied to our task. To the best of our knowledge, only one existing work [32] proposed a compression model for KGE. However, it was based on product quantization, which was not powerful enough to handle complex relations in knowledge graphs. We adopt this method as a baseline in the experiments and show that the proposed LightKG has better performance.

Another related problem is how to sample informative negative samples. In [5, 40], this problem was defined, and the authors designed a generative adversarial learning framework to draw negative samples. However, the computation complexity of this framework is high. In a recent work [36], a self-adversarial sampling scheme RotatE was proposed to re-weight negative samples. In our experiments, we also show that the proposed dynamic negative sampling strategy outperforms RotatE.

3 METHODOLOGY

3.1 Problem Formulation

A knowledge graph (KG) can be represented as $\mathcal{G} = \{(h, r, t)|h, t \in \mathcal{E}, r \in \mathcal{R}\}$, where $\mathcal{E}$ represents the entity set and $\mathcal{R}$ represents the relation set. We denote the number of entities as n_e and the number of relations as n_r. The goal of knowledge graph embedding (KGE) is to learn vector embedding $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{R}^d$ and $\boldsymbol{r} \in \mathbb{R}^d$. For a triple (h, r, t), KGE methods also define a relevance score based on the embeddings $(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$, which quantifies how plausible the triple exists in the graph. Existing methods usually adopt inner product [45] or Euclidean distance [4] based on the embeddings as the scoring function, which can be written as $f(\cdot) : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$. In this paper, we aim to design a lightweight KGE framework, which reduces the required space and speeds up inference.

3.2 Overview

LightKG is an end-to-end framework, which learn quantized entity embedding instead of continous-vector embeddings. Given an entity embedding, we first split the embedding into multiple subspaces and then learn the codeword (Section 3.3) that is the most similar to the entity embedding in the i-th subspace for embedding compression. To preserve the information of embbeding, we propose a novel residual module (Section 3.4), which can leverage multiple codewords to compress embeddings in a sequential manner. In section 3.5, we introduce the loss function of the proposed framework and the training procedure. To better understand our model, we discuss the advantage of LightKG in Section 3.6. Finally, we propose an effective negative sampling method, namely dynamic negative sampling, to make up for the performance drop brought by compression in Section 3.7.

3.3 Codeword Learning

The quantization methods divide a large set of data vectors into groups and each group is represented by its centroids, which are refereed to as codewords. Existing methods usually learn codewords via KMeans [12, 20], which is not differentiable. Thus, we cannot directly adopt existing codeword learning methods in our end-to-end framework. Specifically, the non-differentiabl issue is attributed to the non-differentiable maximum selection operator arg max in the codeword learning procedure.

To enable codeword learning in LightKG, we propose to approximate the maximum selection operator. Moreover, we adopt multiple codebooks instead of one to preserve the information of original embedding as much as possible. The details of including multiple codebooks are introduced in the next subsection.

Formally, we divide an entity embedding $\boldsymbol{e}$ with d dimensions into m subspaces. The divided entity embedding in the i-th subspace is denoted as $\boldsymbol{e}^i \in \mathbb{R}^{d/m}$ and we can represent the original entity embedding as $\boldsymbol{e} = [\boldsymbol{e}^1, \boldsymbol{e}^2, ..., \boldsymbol{e}^m]$. We use B codebooks in each subspace, each of which contains W codewords. Consider the b-th codebook in the i-th subspace, the process of learning $w^b(i)$-th codeword can be represented as matrix multiplication, i.e. $\boldsymbol{c}^b_{w^b(i)}(i) = C^b(i)\boldsymbol{o}$, where $C^b(i) \in \mathbb{R}^{d/m \times W}$ is the b-th codebook in the i-th subspace, and $\boldsymbol{o}$ is a one-hot vector with $\boldsymbol{o}_{w^b(i)} = 1$. Thus, to overcome the obstacle imposed by non-differentiable maximum selection operator, we need to approximate the one-hot vector $\boldsymbol{o}$, which can be loosened by tempered softmax [29]. Specifically, the one-hot vector $\boldsymbol{o}$ can be approximated via $\hat{\boldsymbol{o}}$: $o_j \approx \hat{o}_j = \frac{\exp(s(\boldsymbol{e}^i, \boldsymbol{c}^b_j(i))/T)}{\sum_{j'} \exp(s(\boldsymbol{e}^i, \boldsymbol{c}^b_{j'}(i))/T)}$, where T is the temperature and $s(\cdot)$ is a similarity function. When $T \rightarrow 0$, $\hat{\boldsymbol{o}}$ equals to $\boldsymbol{o}$. Therefore, the codeword selection can be expressed as $\boldsymbol{c}^b_{w^b(i)} \approx C^b(i)\hat{\boldsymbol{o}}$. Following [3], we apply Straight-Through Estimator and rewrite $\boldsymbol{o}$ as $\boldsymbol{o} = \hat{\boldsymbol{o}} + \text{stop\_gradient}(\boldsymbol{o} - \hat{\boldsymbol{o}})$, where stop_gradient indicates the operation to stop the calculation of this gradient. The one-hot vector $\boldsymbol{o}$ is used in the forward propagation while $\hat{\boldsymbol{o}}$ is used in the back propagation to approximately calculate gradients. Based on tempered softmax relaxation and Straight-Through Estimator, LightKG is able to learn codebooks stably in an end-to-end fashion.

The success of codeword selection also relies on the similarity function $s(\cdot)$. The similarity function based on Euclidean distance is usually applied for the codeword selection [20]. However, KGE is not always in the Euclidean space [6], thereby making Euclidean distance based similarity function inappropriate for our problem. Hence, we do not make any assumption on the Euclidean space and propose to learn the similarity function from data [10, 25, 45], which can be parameterized as $s(\boldsymbol{e}, \boldsymbol{c}) = \boldsymbol{e}^T \boldsymbol{M} \boldsymbol{c}$, where $\boldsymbol{M}$ is the learnable parameter. It can be considered as a generalized bilinear attention function [21], which has shown to be effective in capturing data similarity.

3.4 Residual Module

In the previous section, we introduced the component that conduct codeword learning. In this section, we present how to use B codewords to get quantized embedding. The essence of LightKG is that we split embeddings by subspaces and quantize each slice by the selected codeword which is the most similar to the entity embedding. Then the embedding of an entity in a subspace can be obtained by a sum of the selected codewords among B codebooks. However, if we simply quantize the embedding slice for B times to

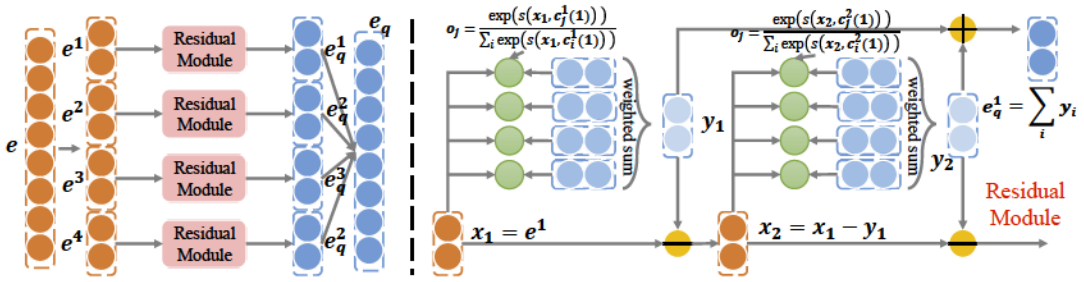


Figure 2: The framework of LightKG.

get B codewords and add the codewords to represent the quantized embedding slice, then the derived codebooks tend to become similar to each other, and there are no additional benefits brought by incorporating more codebooks. Thus, it is crucial to encourage the diversity among codebooks.

One potential strategy is to impose some diversity regularizers (e.g., squared Frobenius norm, von Neumann divergence [22, 44], cosine similarity [2, 43, 46]) into codebook computation. However, this direct incorporation of diversity regularizers may result in two issues. The first is that the regularizers can be satisfied by simply changing the order of codewords in codebooks. Consider an example in which we have B codesbooks that have exactly the same set of codewords, but we simply change the order of the codewords. Then these codebooks are considered to be diverse by the diversity regularizers, but apparently such codebooks do not contain different codewords and the selected codewords may be the same. Second, the diversity regularizers need to be calculated between each pair of codebooks in the same subspace, resulting in $\frac{mB(B-1)}{2}$ regularizer terms. Given $m = 10$ and $B = 16$, 1,200 regularizer terms need to be calculated and the corresponding computation cost is prohibitively high.

Inspired by [8], we propose a residual module to learn diverse codebooks in each subspace. The residual module generate codebooks in a sequential way so that the residual information is learnt from the previous steps. The details of this module is shown in Figure 2. Formally, consider the i-th subspace of entity embedding $\boldsymbol{e}_k$. $Q(x; C^b(i)) : \mathbb{R}^{d/m} \rightarrow \mathbb{R}^{d/m}$ denotes the function to quantize x with codebook $C^b(i)$, i.e. $c^b_{w^b(i)} = Q(x; C^b(i))$. The sequential process can be described as the following recursive relation:

$$x_{b+1} = x_b - Q(x_b; C^b(i)), x_1 = \boldsymbol{e}^i_k. \quad (1)$$

According to Eqn. 1, the input of the quantization function $Q(\cdot)$ is the residual error between the last input of $Q(\cdot)$ and the last quantization results. Therefore, the input of $Q(\cdot)$ is different each time and we can avoid the generation of a trivial solution that involves similar codewords among codebooks, ensuring the diversities among codewords. For each divided entity embedding, we can obtain its corresponding quantized embeddings through the residual module and then we concatenate all the quantized embedding of divided entity embeddings to get our final quanzited embedding $\boldsymbol{e}_q$, as shown in Fig. 2.

3.5 Loss Function

Now that the components are described, we summarize the proposed LightKG framework and training process. LightKG takes the triples in the knowledge graph as input and output the quantized embeddings of entities. The entire framework can be trained in an end-to-end manner. To train LightKG, we first generate the negative sample set $\mathcal{G}'$ following KGE methods [4] by corrupting the observed triples in the knowledge graph. We only corrupt tail entities in triples in this paper. The training of KGE method is usually based on margin-based loss [4] or cross entropy loss [10]. Here we use margin-based loss as an example to present the overall loss defined for LightKG. We use Θ to denote all the parameters to be learned and the objective function of LightKG based on margin loss can be written as follows:

$$\begin{aligned}\mathcal{L}_{LightKG} &= \sum_{(h,r,t)\in\mathcal{G},(h,r,t')\in\mathcal{G}'} \ell(h, r, t, t' \,|\Theta) \\ &= \sum_{(h,r,t)\in\mathcal{G},(h,r,t')\in\mathcal{G}'} [f(\boldsymbol{h}_q, \boldsymbol{r}, \boldsymbol{t}_q) - f(\boldsymbol{h}_q, \boldsymbol{r}, \boldsymbol{t}'_q) + \gamma]_+, \quad (2)\end{aligned}$$

where $\boldsymbol{h}_q, \boldsymbol{t}_q, \boldsymbol{t}'_q$ denotes the quantized entity embedding, $[x]_+$ denotes the positive part of x and $\gamma > 0$ is the margin. The loss function can be easily optimized via gradient descent methods like SGD and ADAM. In the following sections, we drop Θ for simplicity and refer to our loss function as $\sum_{(h,r,t)\in\mathcal{G},(h,r,t')\in\mathcal{G}'} \ell(h, r, t, t')$.

3.6 Advantages of LightKG

We first justify the necessity of adopting LightKG, an end-to-end framework, for memory and inference efficiency. Although there exist some possible solution that could save space and accelerate inference, the accuracy in querying or inference may need to be sacrificed. One possible solution is to first learn entity embedding and then apply quantization-based methods like PQ [20] and OPQ [12] to learn codebooks and codewords. In contrast to LightKG, this is a post-compression approach as the compression is conducted only after entity embedding is learnt. The quantization error of this solution tends to be amplified when computing scores based on Euclidean distance, as proved in the following Proposition 1. We also give an example that shows the possible error made by this solution. In addition, [24] points out that PQ and OPQ are based on Euclidean distance and are incompatible with inner product-based scoring function. Therefore, they are not suited for KGE, motivating us to seek a way to to learn codewords and codebooks via an end-to-end mechanism for embedding compression.

Proposition 1. *Directly applying quantization to entity embedding may amplify error.*

Proof. Suppose head embedding is $\boldsymbol{h}$, tail embedding is $\boldsymbol{t}$. And their corresponding quantization embeddings are $\hat{\boldsymbol{h}}$ and $\hat{\boldsymbol{t}}$. $sup\{||\boldsymbol{h} - \hat{\boldsymbol{h}}||_2\} = \delta$. Consider the TransE score error

$$||||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}||_2 - ||\hat{\boldsymbol{h}} + \boldsymbol{r} - \hat{\boldsymbol{t}}||_2||_2 \leq ||\boldsymbol{h} - \boldsymbol{t} - (\hat{\boldsymbol{h}} - \hat{\boldsymbol{t}})||_2 \leq 2\delta$$

Here the equation can be achieved. And the error of TransE score is twice the error of quantization. □

Next, we discuss the gains in storage and speed savings obtained by using this LightKG framework for entity embedding. We consider the downstream task of querying or completing a knowledge graph. Given a query entity $q \in \mathbb{R}^d$, the objective is to rank all the entities according to the relevance to the query. According to LightKG embeddings, this relevance score for entity e_k can be computed as:

$$\langle q, e_k \rangle = \sum_{i=1}^{m} \sum_{b=1}^{B} \langle q^i, c^b_{w^b_k(i)}(i) \rangle, \text{ or} \quad (3)$$

$$||q - e_k||_2^2 = \sum_{i=1}^{m} ||q^i||_2^2 + \sum_{i=1}^{m} ||e^i_k||_2^2 - 2 \sum_{i=1}^{m} \sum_{b=1}^{B} \langle q^i, c^b_{w^b_k(i)}(i) \rangle. \quad (4)$$

As can be seen, Eqn. 3 and Eqn. 4 define the relevance scores based on the inner product and Euclidean distance respectively, which could correspondingly be applied to translation-based and bilinear models. Then, let us examine the storage savings. Besides space needed to store the codebooks, we only need some space to store the following information. If Eqn. 3 is adopted, then only codeword indices in each subspace need to be stored, i.e. $[w^1_k(1), w^2_k(1), ..., w^B_k(m)] \in \{1, ..., W\}^{Bm}$; If Eqn. 4 is adopted, only codeword indices in each subspace and $\sum_{i=1}^{m} ||e^i_k||_2^2$ (the norm of entity embeddings) should be stored. Therefore, the total memory cost of n_e entities can be reduced from $4n_ed$ Bytes (using 32 bit float representation) to $4BWd + \frac{1}{8}n_emB\log W$ (inner product) or $4BWd + \frac{1}{8}n_emB\log W + 4n_e$ (Euclidean distance), where $4BWd$, $\frac{1}{8}n_emB\log W$, and $4n_e$ are the cost of storing codebooks, indices, and entity embedding norms respectively. Usually n_e is much bigger than BW, so the compression ratio is around $\frac{32d}{mB\log W}$. For example, if we set the number of subspaces as 10, the number of codebooks in each subspace as 16 and the number of codewords in each codebook as 32, the memory cost is only around $\frac{1}{8}$ of the memory needed to store a continuous representation for each entity.

Furthermore, LightKG encoding can greatly reduce query or inference time. If LightKG is not used, we need to score all the entities for a query, so it costs $O(n_ed)$ time. When LightKG is used, then based on Eqn. 3 and Eqn. 4, the computation complexity can be reduced to $O(n_e + BWd)$. The reason is that the scores between q^i and each codeword can be computed as the inner product between the query vector and codeword according to Eqn. 3 and Eqn. 4, and the number of codewords is far less than the number of all entities. When n_e is large, the speedup ratio is also large. Besides, the querying process involve the searching for the top K entities with the highest relevance scores. This is known as nearest neighbour search or maximum inner product search. When LightKG is used, this searching process can be coordinated with the inverted file system practically to further improve the inference efficiency. We discuss the computation complexity of combining inverted file system with LightKG in Section 4.1.

3.7 Dynamic Negative Sampling

Preparing the training set is an important step for KGE. The key is to obtain high-quality negative samples. Hence, to further improve LightKG, we propose a quantization-based dynamic negative sampling method named DSLightKG, that could improve sampling efficiency and quality. This sampling strategy can not only work with LightKG, but also apply to other KGE methods. In the following, we use "DSX" to represent applying dynamic negative sampling to KGE method X.

Algorithm 1: Dynamic Negative Sampling

Input: All corrupted candidate entities: $\{t_i | (h, r, t_i) \notin \mathcal{G}\}$; K for top-K entities.
Output: top-K entities.

Seperate corrupted candidate entity embeddings into two parts
  $T^1 = [t_1[1 : d/2], ..., t_{|\mathcal{E}'|}[1 : d/2]]$ and
  $T^2 = [t_i[d/2 + 1 : d], .., t_{|\mathcal{E}'|}[d/2 + 1 : d]]$.
Apply KMeans to T^1 and T^2 separately. Then get cluster centroid C^1 corresponding to T^1 and C^2 corresponding to T^2.
Compute the distance $d^1 = -f(h[1 : d/2], r[1 : d/2], C^1)$ and $d^2 = -f(h[d/2 + 1 : d], r[d/2 + 1 : d], C^2)$.
Sort d^1 and d^1 in descending order.
Output = []; Visited[1:len(d^1),1:len(d^2)]=0; pqueue=PriorityQueue.
pqueue.push((1, 1), $d^1[1] + d^2[1]$)
while *len(Output)*<K **do**
  $((i, j), d)$=pqueue.pop()
  Visited$[i, j] = 1$
  Output.append((i, j))
  if $i < len(d^1)$ **then**
    if $j == 1$ *or Visited*$[i + 1, j - 1] == 1$ **then**
      pqueue.push($(i + 1, j)$, $d^1[i + 1] + d^2[j]$)
  if $j < len(d^2)$ **then**
    if $i == 1$ *or Visited*$[i - 1, j + 1] == 1$ **then**
      pqueue.push($(i, j + 1)$, $d^1[i] + d^2[j + 1]$)
return entity ids corresponding to Output.

Given a positive triple (h, r, t), we denote its corresponding corrupted set as $\{t_1, ..., t_m\}$ (negative triples). According to Section 3.5, the loss of the given triple is $\frac{1}{m}\sum_{i=1}^{m} \ell(h, r, t, t_i)$, which is lower when the positive triple has a high relevance score than that of the negative triple and vice versa. Here, t_i is sampled from $\{t' | (h, r, t') \notin \mathcal{G}\}$ via uniform sampling. However, most of the randomly generated negative samples are naive and tend to be less informative. To further improve the sampling efficiency, we propose to increase the weights of informative negative samples. In this paper, we use the relevance score to distinguish the informative and uninformative negative samples. The intuition is that if a negative sample is not easy for the model to identify, this negative sample is informative for the model at this stage. In KGE framework, we use Euclidean distance or inner product to define the triple score, i.e. given a triple (h, r, t), its score is defined as $f(h, r, t) = -||h+r-t||_2$ or $\langle h, r, t \rangle$. For a negative sample (h, r, t_i), we denote its score as f_i for short. A negative sample with a high score indicates that it is informative. Thus, we propose to sample negative samples according to the distribution as $P_I(i) = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$.

However, P_I needs to compute the partition function $\sum_j \exp(f_j)$, whose time complexity is $O(n_ed)$. Since the complexity of computing P_I is prohibitively high, we alternatively seek to approximate $P_I(i)$ by probability $P_D(i)$ based on top-K entities as follows:

$$P_D(i) = \frac{1}{2}(P_D^{top-K}(i) + P_D^U(i)), \quad (5)$$

where $P_D^{top-K}(i) = \begin{cases} \frac{1}{K} & i \in \text{top-K entities} \\ 0 & otherwise \end{cases}$, $P_D^U(i) = \frac{1}{|\mathcal{E}'|}$.
Here, top-K entities are refereed to as the K entities with the highest scores according to $\exp(d_i)$. The following proposition shows $P_D(i)$ is closer to $P_I(i)$ than discrete uniform distribution.

Proposition 2. *When* $\sum_{i \in \mathcal{T}} P_I(i) \geq log_{1+a}2$, $KL(P_I||P_D) \leq KL(P_I||P_U)$ *holds, where* P_D *is the distribution of dynamic negative*

sampling, P_U is the distribution of discrete random sampling, P_I is the ideal negative distribution, $K = |\mathcal{E}^{'}|/a$ and τ is $\{i|i$ is top-K entities$\}$.

Proof. Recall that $P_E(i) = \frac{1}{|\mathcal{E}'|}$, $P_D(i) = \frac{1}{2}(\frac{1}{K} + \frac{1}{|\mathcal{E}'|})$ if $i \in$ top-K entities and $P_D(i) = \frac{1}{2|\mathcal{E}'|}$ otherwise, and $P_I(i) = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$. In the following, $i \in \mathcal{T}$ denotes $\{i \in$ top-K entities$\}$. Therefore we have $KL(P_I||P_U) = -\sum_i P_I(i)\ln(\frac{\frac{1}{|\mathcal{E}'|}}{P_I(i)})$, and $KL(P_I||P_D) = -\sum_{i\in\mathcal{T}} P_I(i)\ln(\frac{\frac{1}{2}(\frac{1}{K}+\frac{1}{N})}{P_I(i)}) - \sum_{i\notin\mathcal{T}} P_I(i)\ln(\frac{\frac{1}{2|\mathcal{E}'|}}{P_I(i)})$. Then we have

$$KL(P_I||P_U) - KL(P_I||P_D)$$
$$= \sum_{i\notin\mathcal{T}} P_I(i)\ln(\frac{1}{2}) + \sum_{i\in\mathcal{T}} P_I(i)\ln\frac{|\mathcal{E}'|+K}{2K} = \ln(\frac{1}{2}) + \sum_{i\in\mathcal{T}} P_I(i)\ln(\frac{|\mathcal{E}'|}{K}+1)$$

Because $\ln(\frac{1}{2}) + \sum_{i\in\mathcal{T}} P_I(i)\ln(\frac{|\mathcal{E}'|}{K}+1) = \ln(\frac{1}{2}) + \sum_{i\in\mathcal{T}} P_I(i)\ln(1+a) \geq 0$, we have $KL(P_I||P_U) \geq KL(P_I||P_D)$. Because softmax focus on the largest value in one vector and $a = |\mathcal{E}^{'}|/K$ is a large number, the condition $\sum_{i\in\mathcal{T}} P_I(i) \geq \log_{1+a}2$ is easy to be satisfied. □

Although $P_D^{top-K}(i)$ can approximate P_I, it is still challenging to find an way to locate the top-K entities with a low computational complexity. If we exhaustively compute all the scores and sort them to get the top-K entities, the computation complexity is $O(n_e d + n_e \log K)$, which is even higher than that of computing P_I. Inspired by [1], $P_D^{top-K}(i)$ can be approximately computed efficiently via quantization. Given a positive triple (h, r, t) and its negative sample $(h, r, t^{'})$, we first divide entity embeddings T into two subspace T^1 and T^2. Then we apply KMeans to T^1 and T^2 respectively to obtain their corresponding centroids C^1 and C^2. The relevance score is then calculated between h, r and C^1, C^2 respectively. The approximate top-K high-quality negative samples can be located when we find K entities whose corresponding two centroids both have high relevance scores based on h and r. The whole process is summarized in Algorithm 1. The computation complexity of dynamic sampling is $O((kd + 2k\log k + K\log K)n + n_e dkt)$, where n is the number of training data. It is much less than the complexity of exhaustive computing $n_e dn$ since $kt \ll n$ and $k \ll n_e$. More details about the complexity of dynamic negative sampling are in Section 4.1.

4 COMPLEXITY ANALYSIS

4.1 Search with Inverted Index File Complexity

In this section, we discuss the search complexity with inverted file index, which is mentioned in Section 3.6. Usually, nearest neighbour search is coordinated with the inverted file to prevent an exhaustive search of all of the vectors. We first apply KMeans with k cluster centroids to entity embeddings. When conducting approximate top-K nearest neighbour search, we get $\hat{K}$ ($\hat{K} > K$) approximate nearest neighbours via KMeans, and then seach K nearest neighbours from $\hat{K}$ candidates via LightKG. Therefore, its time complexity is $O(kd + k\log k + \hat{K} + BWd)$, where $kd + k\log k$ is the cost of searching via KMeans and $\hat{K} + BWd$ is the time complexity of LightKG. When n_e is very huge, $kd + k\log k$ can be much less than n_e.

4.2 Dynamic Negative Sampling Complexity

Firstly, we analyze the complexity of negative sampling without approximation (sampling from distribution P_I in Section 1). For one triple (h, r, t), it needs to compute the distance between $\boldsymbol{h} + \boldsymbol{r}/\boldsymbol{h} * \boldsymbol{r}$ and all entities, which costs $O(n_e d)$. Therefore, totally it costs $O(n_e dn)$, where n is the number of train data.Then we consider the complexity of the dynamic negative sampling. For the KMeans step, it costs $O(n_e dkt)$ [15], where t is the iteration times and k is the number of clusters. For computing the distance between $\boldsymbol{h} + \boldsymbol{r}/\boldsymbol{h} * \boldsymbol{r}$ and centroids, it costs $O(dk)$. Then sorting the distance costs $O(2k\log k)$. Suppose we want to search top-K closest distance given two sorted lists, it costs $O(K\log K)$ via priority queue structure. Therefore, totally dynamic negative sampling costs $O((kd+2k\log k+K\log K)n+n_e dkt)$. Because usually the number of training data and entities is huge, we have $kt \ll n$ and $k \ll n_e$. Thus, the complexity of dynamic negative sampling is much less than the complexity of negative sampling without approximation.

Table 1: Statistics of datasets.

Dataset	#entities	#relations	#Train	#Validation	#Test
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134
FB15K	14,951	1,345	483,142	50,000	59,071
FB15K237	14,541	237	272,115	17,535	20,466
YAGO3	123,182	37	1,079,040	5,000	5,000

5 EXPERIMENTS

In this section, we evaluate the proposed LightKG model and dynamic negative sampling method with the goal of answering the following questions. 1) How does LightKG perform as compared to state-of-the-art baselines? 2) Is the proposed dynamic negative sampling effective and efficient in enabling the learning of better embeddings? 3) Is the proposed LightKG able to extend to deep learning-based KGE method? 4) Is the proposed residual module effective in the learning of quantization model? 5) How does the performance change with respect to different parameters? 6) How does LightKG improve the search and memory efficiency?

5.1 Datasets and Experiment Settings

5.1.1 Datasets. We use four public benchmark datasets including WN18 [4], WN18RR [37], FB15K [4] and FB15K237 [10] to evaluate the proposed algorithm. Additional, we use a large-scale dataset YAGO3 [35] to show the efficiency of the proposed model. We summarize the statistics of all the datasets in Table 1.

5.1.2 Baselines. We compare proposed LightKG and DSLightKG (LightKG with dynamic sampling), with the following baselines: TransE [4], ConvE [10], DistMult [45], OPQ [12], RVQ [8], and TS-CL [32]. TransE and DistMult are representative translation-based and bilinear KGE methods respectively. They provide the ceiling performance for quantization models. OPQ, RVQ and TS-CL are three quantization models. OPQ, RVQ are two post-compression methods. TS-CL is an end-to-end quantization method.

5.1.3 Evaluation Metric and Implementation Details. To evaluate the proposed model, we follow [36] and use standard evaluation metrics Mean Reciprocal Rank (MRR) and Hits at N (Hits@N) in the filtered setting. For both metrics, the higher the better. We rank triples in the test set against all other generated candidate triples which are not in the training, validation, and test set. In this paper, we focus on the task of predicting the tail entity given a specific relation and a head entity, i.e., $(h, r, ?)$. We implement LightKG

based on OpenKE [14], an open source framework for KGE. We set $d = 200$, $m = 10$, $B = 16$, and $W = 32$ in experiments. For LightKG, we draw one negative sample for each positive sample. For DSLightKG and DSTransE, we sample nine negative samples from uniform sampling and one negative sample from dynamic negative sampling. K in Eqn. 5 is set to be 100. To accelerate the convergence of LightKG, we add one regularizer in Eqn. 2 following [32]: $||\boldsymbol{e}_q - \boldsymbol{e}||$, where $\boldsymbol{e}$ is the entity embedding before quantization and $\boldsymbol{e}_q$ is the entity embedding after quantization. We tune its coefficient and learning rate on the validation set via grid search over {1e-5,1e-4,1e-3,1e-2}.

Table 2: Comparison with baselines on the four datasets. TransE and DistMult provides the ceiling performance for tasks respectively. The highest scores per category are bold.

Category	Method	WN18			WN18RR		
		H@1	H@10	MRR	H@1	H@10	MRR
Translation	TransE	12.54	89.78	41.65	1.91	46.20	17.47
	OPQ	7.30	64.26	27.52	1.56	31.94	11.78
	RVQ	6.76	64.84	27.21	1.47	32.00	11.71
	TS-CL	2.92	66.20	24.69	0.54	29.71	10.56
	LightKG	**8.42**	**89.10**	**38.91**	**1.66**	**45.31**	**16.95**
Bilinear	DistMult	67.71	93.92	78.73	41.07	47.13	43.32
	OPQ	28.12	56.64	37.77	17.29	31.11	21.98
	RVQ	15.70	46.10	25.50	5.39	20.39	10.35
	TS-CL	42.36	67.58	51.07	22.27	32.61	25.94
	LightKG	**67.10**	**93.36**	**78.28**	**40.65**	**46.39**	**42.73**
Category	Method	FB15K			FB15K237		
		H@1	H@10	MRR	H@1	H@10	MRR
Translation	TransE	25.12	65.56	39.25	20.93	46.66	29.49
	OPQ	15.88	48.36	26.67	**17.70**	41.78	23.91
	RVQ	13.23	44.64	23.60	15.93	42.11	24.75
	TS-CL	17.99	47.17	28.07	12.04	36.11	20.43
	LightKG	**25.24**	**65.45**	**39.24**	16.70	**42.88**	**25.62**
Bilinear	DistMult	42.08	82.74	56.45	21.86	48.12	30.50
	OPQ	13.14	42.17	22.71	15.89	44.95	24.79
	RVQ	9.15	34.52	17.54	**20.22**	46.34	**28.94**
	TS-CL	16.62	49.04	27.39	14.31	40.89	23.22
	LightKG	**40.97**	**82.09**	**55.47**	19.69	**47.57**	28.84

5.2 Comparison with Baselines

In this section, we report the performance of baselines and the proposed LightKG in Table 2 to answer the first question and present our findings.

First, LightKG outperforms other quantization methods dramatically. For quantizing TransE, the proposed LightKG has about 26.2% improvement concerning Hits@10 on average compared with OPQ, RVQ and TS-CL; for quantizing DistMult, the proposed LightKG has about 35.5% improvement with respect to Hits@10 on average compared with the quantization baselines. Therefore, LightKG is much more effective when learning lightweight KGE compared with existing quantization methods.

Second, comparing the performance of LightKG with respect to that of TransE and DistMult which provide the ceiling performance, the gap is not large. Because lightweight methods use much fewer parameters than the original KGE methods that output continuous representations, their performance may not be as good as that of the KGE methods. Therefore, the goal of LightKG is to reduce the gap between itself with TransE and DistMult in terms of performance. Compared with TransE, LightKG only drops about 2.75% accuracy with respect to Hits@10 on average; compared with DistMult, LightKG only drops about 1.03% accuracy in terms of Hits@10 on average. Thus, LightKG can preserve the model's performance with less parameters.

Third, the performance of OPQ and RVQ depends on if the embeddings form a spherical shape in the low-dimensional space, but LightKG does not have this restriction. OPQ and RVQ are based on KMeans. However, KMeans is sensitive to data distribution. We record their reconstruction errors of KMeans on FB15K and FB15K237. The reconstruction error of KMeans on FB15K237 is only 0.0069 while reconstruction error of KMeans on FB15K is 0.128. So OPQ and RVQ have better performance on FB15K237 than on FB15K. Therefore, the two methods highly depend on data distribution. In contrast, because we choose a more general similarity function $s(\cdot)$ instead of Euclidean distance for LightKG, LightKG can handle a wider range of data distributions and thus achieve much better performance.

Forth, Table 2 shows that the proposed model achieves more improvements on WN18RR than that on FB15K237. Such an observation may indicate that the proposed model LightKG performs better with a dense knowledge graph, where more context information is available for each entity. The similar observation is also shown in a contemporary work [26] for recommendation quantization task.

Table 3: Comparison with different negative sampling methods.

	FB15K237				FB15K			
	Hits@1	Hits@10	MRR	IMP(%)	Hits@1	Hits@10	MRR	IMP(%)
TransE	20.93	46.66	29.49	37.23	25.12	65.56	39.25	17.30
UTransE	27.59	58.62	38.06	6.33	23.68	71.65	42.08	9.41
TransE-adv	28.72	59.39	39.11	3.48	25.78	74.83	45.14	1.99
DSTransE	30.57	60.03	40.47	-	29.33	72.59	46.04	-
LightKG	16.70	42.88	25.62	13.43	25.24	65.45	39.24	11.77
ULightKG	16.86	43.77	25.65	13.29	30.75	67.69	43.62	0.55
LightKG-adv	18.10	44.92	27.07	7.35	30.70	67.90	43.60	0.60
DSLightKG	20.74	46.63	29.06	-	31.07	67.58	43.86	-

5.3 Dynamic Negative Sampling

In this section, we will answer the second question. We show the results of different negative sampling methods on FB15K and FB15K237 in Table 3. In the table, TransE and LightKG sample one negative sample via uniform sampling; UTransE and ULightKG sample ten negative samples via uniform sampling for TransE and LightKG respectively; TransE-adv and LightKG-adv apply the state-of-the-art adversary negative sampling method [36] to TransE and LightKG and sample ten negative samples respectively; DSTransE and DSLightKG apply the proposed dynamic negative sampling to TransE and LightKG to sample ten negative samples respectively. We record the running time of different negative sampling methods to show the effectiveness of dynamic negative sampling. Based on these results, we have the following findings.

First, dynamic sampling can improve model accuracy greatly. The performance of DSTransE and DSLightKG is much better than that of UTransE and ULightKG respectively on two datasets. DSTransE has 7.87% improvement on average compared with UTransE; DSLightKG has 6.92% improvement averagely compared to ULightKG

and it achieves competitive results compared with TransE. It shows the effectiveness of the proposed sampling strategy that introduces more informative negative samples. As for adversary sampling, it uses adaptive weights for different negative samples, but the sampling is still conducted based on uniform sampling, making it difficult to get informative samples in this way. Therefore, this baseline performs better than uniform sampling, but its performance is inferior compared with the proposed dynamic sampling strategy.

Second, sampling more negative samples can improve model performance. On two datasets, UTransE and ULightKG both have significant improvements compared with TransE and LightKG respectively. The possible reason is that more negative samples provide more information between entities and relations, which helps the learning of better representations.

Third, dynamic negative sampling is much more efficient than exhaustive sampling. The running times of uniform, dynamic and exhaustive sampling for one batch are 116.3, 519.7 and 1299.1 seconds respectively. Dynamic negative sampling only costs less than half of the time by exhaustive sampling. It also validates the time complexity analysis in Section 4.1.

5.4 Extension to Deep Models

Deep learning-based KGE methods have attracted more and more attention in recent years. Therefore, in this section, we study whether the proposed LightKG can be applied to deep learning-based KGE methods to answer the third question. We choose the state-of-the-art deep model ConvE [10] as our baseline and apply LightKG to quantize it. The results of ConvE and LightKG are shown in Table 4.

According to Table 4, LightKG is effective when applied to quantize ConvE. The gap between ConvE and LightKG in terms of performance is small, i.e., the MRR of LightKG only drops 2.57% with respect to MRR compared with ConvE. It shows that LightKG is a general framework which can also be applied to deep learning based KGE methods to improve the efficiency with only very limited drop in accuracy.

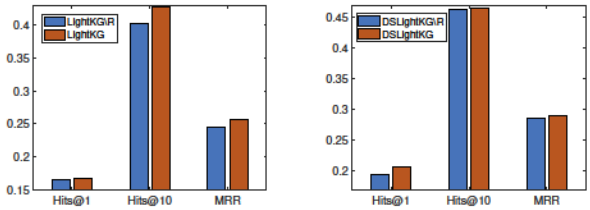
Table 4: The results of ConvE and LightKG on FB15K237.

	Hits@1	Hits@10	MRR	GAP(%)
ConvE	21.92	48.52	30.71	-
LightKG-ConvE	21.38	47.04	29.92	-2.57

5.5 Residual Mechanism

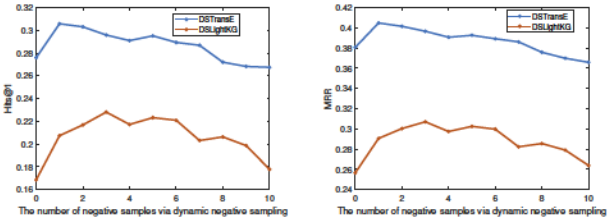
In this section, we demonstrate an ablation study with respect to the proposed residual module to answer the forth question. We show the results of the model with and without residual module on FB15K237 dataset in Fig. 3.

According to Fig. 3(a), the residual mechanism significantly improves the model's performance. Compared with LightKG\R (the model without the residual module), LightKG has 1.2% improvement, 6.4% improvement and 4.7% improvement with respect to Hits@1, Hits@10 and MRR respectively. This illustrates the benefit of adopting the residual module to learn high-quality codebooks. Similar patterns can be observed on LightKG with dynamic negative sampling. Compared with DSLightKG\R, according to Fig. 3(b), DSLightKG has 6.9% improvement with respect to Hits@1 as well. It achieves less improvement compared with LightKG on average. A possible reason is that both DSLightKG and DSLightKG\R use



(a) (b)

Figure 3: The results of the model with residual module and without residual module on FB15K237. Fig. 3(a) shows the results LightKG and LightKG without residual module (LightKG\R); Fig. 3(b) is DSLightKG and DSLightKG without residual module (DSLightKG\R).



(a) (b)

Figure 4: The results of DSTransE and DSLightKG with different number of negative samples sampled by dynamic negative sampling on FB15K237.

dynamic negative sampling, which somewhat mitigate the issue with DSLightKG\R.

Table 5: The results of DSLightKG adopting different depth of residual layers on FB15K237.

	Hits@1	Hits@10	MRR	GAP(%)
TransE	20.93	46.66	29.49	-
512×1 coding	18.01	41.49	25.95	-12.00
256×2 coding	21.97	46.11	29.85	+1.22
128×4 coding	21.54	46.91	29.83	+1.15
64×8 coding	22.69	47.60	30.92	+4.85
32×16 coding	20.74	46.63	29.06	-1.46
16×32 coding	21.43	45.75	29.69	+0.68

5.6 Sensitivity w.r.t Depth of Residual Layers

In this section, we study how the depth of residual layers influences model performance to answer the fifth question. To ensure fair comparison, we keep the total number of parameters constant, i.e. the number of codewords in one codebook times the number of codebooks in one subspace remain the same. We show the results in Table 5. In this table, "$i \times j$ coding" means that each codebook consists of i codewords and in each subspace there are j codebooks. j represents the depth of residual layers as well.

According to Table 5, when the depth is greater then one, the performance of DSLightKG is stable with respect to the depth. The performance is very close to the ceiling performance achieved by TransE, and for some depth parameters, the performance is even a bit better than that of TransE. However, when the depth is set to be one, the model performance has a big drop. The reason is that when the depth is one, there is no residual layer in the model, and thus the model degrades into TS-CL. Therefore, this result again supports the effectiveness of the residual module.

5.7 Sensitivity w.r.t The Number of Negative Samples via Dynamic Negative Sampling

In this section, we study how the number of negative samples sampled via dynamic negative sampling influences model performance. We show how Hits@1 and MRR of DSTransE and DSLightKG change with respect to different numbers of negative samples sampled via dynamic negative sampling in Fig. 4.

According to Fig. 4, the model performance and the number of dynamic negative samples are not exactly positively correlated. For DSTransE, 10% negative samples from dynamic sampling and 90% negative samples from uniform sampling achieve the best performance; for DSLightKG, 30% negative samples from dynamic sampling and 70% negative samples from uniform sampling achieve the best performance. On one hand, because hard samples are indistinguishable to the model, sampling too many hard samples makes it too difficult to learn parameters. On the other hand, we hope that dynamic negative sampling can approximate exhaustive sampling but sampling too many hard samples may lead to the deviation of the sampled data from the true distribution.

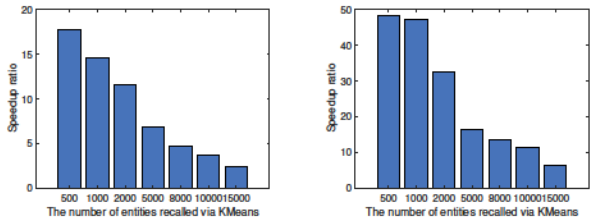
5.8 Efficiency Comparison

In this section, we study the efficiency of the proposed LightKG, including both search efficiency and storage efficiency. We conduct the experiment on WN18 and YAGO3. We apply an inverted file to LightKG since it is a practical setting. We use KMeans to approximate search $\hat{K}$ candidate entities first, and then apply LightKG to search K entities from the $\hat{K}$ candidates. The time for both the two search processes is counted in the total search time. In this study, We adopt the unfilter setting for evaluation to avoid additional operations. We show the speedup ratio in Fig. 5, the accuracy of approximate search in Fig. 6, and the compression ratio in Fig. 7. We observe the following results.

First, recalling more entities from KMeans can improve accuracy but decrease the efficiency of the model. From Fig. 6, we can see that as the the number of recalled entities becomes bigger, the performance of LightKG is closer to DistMult's. However, from Fig. 5, we find that the speedup ratio and the number of entities recalled from KMeans are negatively correlated. This is in line with our intuition. Recall the time complexity of search is $O(kd+k\log k+\hat{K}+BWd)$. Increasing the number of recalled entities corresponds to increasing $\hat{K}$, which leads to an increase in the running time. However, with an increase in the number of recalled entities, more candidates are provided for LightKG, which improves the accuracy. Overall, we can find an equilibrium point where the model has a fast speed with high accuracy. On WN18 dataset, if we choose to recall 10000 entities, the model achieves good performance and is 4 times faster than exhaustive search; on YAGO3, when 5000 entities are recalled, the model has good performance and is about 17 times faster than exhaustive search.

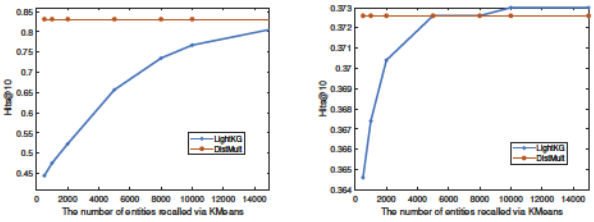
Second, LightKG can save lots of storage space. According to Fig. 7, on WN18 and YAGO3, the compression ratio is more than 7, which also validates the space complexity analysis in Section 3.1.

Third, the advantage of LightKG is larger with more entities. According to Fig. 5, the speedup ratio on YAGO3 is much greater than that on WN18. Note that the search complexity of exhaustive search grows linearly with the number of entities, but in the complexity of LightKG, only the number of codebooks and codewords play a role, which are much smaller than the number of entities. Therefore, the speedup ratio is higher on dataset with more entities.



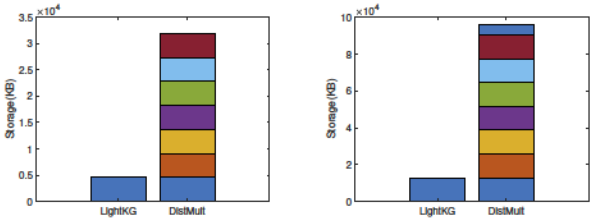
(a) WN18 (b) YAGO3

Figure 5: The speedup ratio of LightKG on WN18 and YAGO3.



(a) WN18 (b) YAGO3

Figure 6: The accuracy of approximate search on WN18 and YAGO3.



(a) WN18 (b) YAGO3

Figure 7: The storage cost of LightKG on WN18 and YAGO3.

6 CONCLUSIONS

In this paper, we propose a lightweight KGE framework, LightKG, for efficient inference and storage. We proposed to divide the space of knowledge graph embeddding into subspace, and in each subspace, we use the combination of codewords within multiple codebooks to represent a slice of the knowledge graph embeddding. This is a general framework, which can be easily plugged into most of the KGE methods including TransE, DistMult and ConvE. To further improve the model performance, we propose a dynamic negative sampling strategy, which can draw informative negative samples efficiently. We conduct extensive experiments on multiple public benchmark datasets to evaluate the proposed LightKG and sampling method. Experimental results show that LightKG has less than 3% accuracy lost while it can obtain over 15x speedup ratio for inference and 7x compression ratio. The dynamic negative sampling method has more than 19% improvement on average.

ACKNOWLEDGEMENT

This work is supported in part by the US National Science Foundation under grant NSF IIS-1747614 and NSF IIS-2141037. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[2] Yebo Bao, Hui Jiang, Lirong Dai, and Cong Liu. [n.d.]. Incoherent training of deep neural networks to de-correlate bottleneck features for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013), 2787–2795.

[5] Liwei Cai and William Yang Wang. 2017. Kbgan: Adversarial learning for knowledge graph embeddings. *arXiv preprint arXiv:1711.04071* (2017).

[6] Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. 2020. Low-Dimensional Hyperbolic Knowledge Graph Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6901–6914.

[7] Xiaojun Chen, Shengbin Jia, and Yang Xiang. 2020. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications* 141 (2020), 112948.

[8] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors* 10, 12 (2010), 11259–11273.

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).

[10] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[11] Laura Dietz, Alexander Kotov, and Edgar Meij. 2018. Utilizing knowledge graphs for text-centric information retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1387–1390.

[12] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[13] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[14] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding. In *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*. 139–144.

[15] John A Hartigan and Manchek A Wong. 1979. AK-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28, 1 (1979), 100–108.

[16] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 105–113.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems* 29 (2016), 4107–4115.

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[19] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[20] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[21] Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. 2018. Bilinear attention networks. *arXiv preprint arXiv:1805.07932* (2018).

[22] Brian Kulis, Mátyás A Sustik, and Inderjit S Dhillon. 2009. Low-Rank Kernel Learning with Bregman Matrix Divergences. *Journal of Machine Learning Research* 10, 2 (2009).

[23] Defu Lian, Qi Liu, and Enhong Chen. 2020. Personalized ranking with importance sampling. In *Proceedings of The Web Conference 2020*. 1093–1103.

[24] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. Lightrec: A memory and search-efficient recommender system. In *Proceedings of The Web Conference 2020*. 695–705.

[25] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2009–2019.

[26] Defu Lian, Xing Xie, Enhong Chen, and Hui Xiong. [n.d.]. Product Quantized Collaborative Filtering. *IEEE Transactions on Knowledge and Data Engineering* ([n. d.]).

[27] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[28] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2018. Entity-duet neural ranking: Understanding the role of knowledge graph semantics in neural information retrieval. *arXiv preprint arXiv:1805.07591* (2018).

[29] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

[30] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data.. In *Icml*, Vol. 11. 809–816.

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.

[32] Mrinmaya Sachan. 2020. Knowledge Graph Embedding Compression. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

[33] Shengtian Sang, Zhihao Yang, Lei Wang, Xiaoxia Liu, Hongfei Lin, and Jian Wang. 2018. SemaTyP: a knowledge graph based literature mining method for drug discovery. *BMC bioinformatics* 19, 1 (2018), 193.

[34] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[35] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. 697–706.

[36] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).

[37] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*. 1499–1509.

[38] Théo Trouillon, Christopher R Dance, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2017. Knowledge graph completion via complex tensor factorization. *arXiv preprint arXiv:1702.06879* (2017).

[39] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 417–426.

[40] Peifeng Wang, Shuangyin Li, and Rong Pan. 2018. Incorporating gan for negative sampling in knowledge representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[41] Yuxiang Wang, Zhangpeng Ge, Haijiang Yan, Xiaoliang Xu, and Yixing Xia. 2019. Semantic locality-based approximate knowledge graph query. *Concurrency and Computation: Practice and Experience* 31, 24 (2019), e5345.

[42] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[43] Pengtao Xie, Yuntian Deng, and Eric Xing. 2015. Diversifying restricted boltzmann machine for document modeling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1315–1324.

[44] Pengtao Xie, Aarti Singh, and Eric P Xing. 2017. Uncorrelation and evenness: a new diversity-promoting regularizer. In *International Conference on Machine Learning*. 3811–3820.

[45] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).

[46] Yang Yu, Yu-Feng Li, and Zhi-Hua Zhou. 2011. Diversity regularized machine. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[47] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.

[48] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.