# TinyKG: Memory-Efficient Training Framework for Knowledge Graph Neural Recommender Systems

Huiyuan Chen
hchen@visa.com
Visa Research
USA

Xiaoting Li
xiaotili@visa.com
Visa Research
USA

Kaixiong Zhou
Kaixiong.Zhou@rice.edu
Rice University
USA

Xia Hu
xia.hu@rice.edu
Rice University
USA

Chin-Chia Michael Yeh
miyeh@visa.com
Visa Research
USA

Yan Zheng
yazheng@visa.com
Visa Research
USA

Hao Yang
haoyang@visa.com
Visa Research
USA

## ABSTRACT

There has been an explosion of interest in designing various Knowledge Graph Neural Networks (KGNNs), which achieve state-of-the-art performance and provide great explainability for recommendation. The promising performance is mainly resulting from their capability of capturing high-order proximity messages over the knowledge graphs. However, training KGNNs at scale is challenging due to the high memory usage. In the forward pass, the automatic differentiation engines (*e.g.*, TensorFlow/PyTorch) generally need to cache all intermediate activation maps in order to compute gradients in the backward pass, which leads to a large GPU memory footprint. Existing work solves this problem by utilizing multi-GPU distributed frameworks. Nonetheless, this poses a practical challenge when seeking to deploy KGNNs in memory-constrained environments, especially for industry-scale graphs.

Here we present TinyKG, a memory-efficient GPU-based training framework for KGNNs for the tasks of recommendation. Specifically, TinyKG uses exact activations in the forward pass while storing a quantized version of activations in the GPU buffers. During the backward pass, these low-precision activations are dequantized back to full-precision tensors, in order to compute gradients. To reduce the quantization errors, TinyKG applies a simple yet effective quantization algorithm to compress the activations, which ensures unbiasedness with low variance. As such, the training memory footprint of KGNNs is largely reduced with negligible accuracy loss. To evaluate the performance of our TinyKG, we conduct comprehensive experiments on real-world datasets. We found that our TinyKG with INT2 quantization aggressively reduces the memory footprint of activation maps with 7×, only with 2% loss in accuracy, allowing us to deploy KGNNs on memory-constrained devices.


*RecSys '22, September 18–23, 2022, Seattle, WA, USA*

ACM ISBN 978-1-4503-9278-5/22/09…$15.00
https://doi.org/10.1145/3523227.3546760

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Collaborative filtering**; • **Software and its engineering** → **Memory management**.

## KEYWORDS

Knowledge Graph Neural Network, Quantization, Tiny Machine Learning, Memory Compression

**ACM Reference Format:**
Huiyuan Chen, Xiaoting Li, Kaixiong Zhou, Xia Hu, Chin-Chia Michael Yeh, Yan Zheng, and Hao Yang. 2022. TinyKG: Memory-Efficient Training Framework for Knowledge Graph Neural Recommender Systems. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3523227.3546760

## 1 INTRODUCTION

Knowledge-aware recommender systems, leveraging the power of Knowledge Graphs (KGs) [1, 45], have recently gained great attention as they achieve state-of-the-art performance in graph-based recommendation [37, 40, 41, 44, 48]. A core benefit of KGs is that they are able to provide high-order connectivity information among items via different types of relations. Such multi-type relations can be seamlessly integrated with user-item interactions, which largely alleviates the data sparsity issues in traditional recommender systems [3, 6, 7, 40, 41, 48]. For instance, CKE [48] combines collaborative filtering with structural knowledge, textual information, and visual signals in a unified framework. Jointly learning the multi-modal heterogeneous graph significantly boosts the quality of recommender systems.

Recently, Knowledge Graph Neural Networks (KGNNs) become one of the most popular graph-based models in recommendation [3, 8, 40–44]. KGNNs use a message-passing mechanism over the KGs, which can be summarized into three stages: i) The input KG is

encoded into an embedding space where each KG entity (*e.g.*, users, items, and attributes) is represented by a low-dimensional vector; ii) Each layer updates the representation of each entity by recursively aggregating and transforming over the representations of its neighbors in the KG; iii) A readout layer is used to obtain the final representation of each entity for the downstream tasks (*e.g.*, link prediction). For example, KGNN-LS [40] transforms the KGs into a user-specific weighted graph and then adopts graph convolution to compute personalized item embeddings with label smoothness regularization. KGAT [41] recursively propagates the embeddings from a node's neighbors to refine the node's embedding and employs an attention mechanism to discriminate the importance of the neighbors. The success of KGNNs shows that capturing high-order proximity messages over multi-hop neighbors in the KGs is essential for the tasks of recommendation.

Despite their promising performance, training KGNNs at scale is still a challenging problem due to its high computational costs and memory footprint. Modern parallel processors (*e.g.*, GPUs) often have limited high bandwidth memory capacity. One straightforward strategy is to reduce the batch size to fit the capacity. Nevertheless, a small training batch size leads to poor compute saturation and may cause side effects for both convergence and accuracy [19, 35].

Alternatively, various distributed training frameworks [24, 49, 50] have been proposed to parallelize the computations across multiple CPU/GPU accelerators for large-scale KGs. For example, DGL-KE [49] and PBG [24] are two popular distributed frameworks for traditional KG embedding models such as TransE [1] and DistMult [45]. These methods treat each *head-relation-tail* triplet in KGs independently so that the input KGs can be partitioned easily and models can be trained in parallel. However, such single-hop distributed frameworks cannot be trivially used for training multi-hop GNN-based KG models. KGNNs require traversing multiple relations in KGs, which span different partitions to learn more complex dependencies among entities [33]. Moreover, distributed KG systems often require high latency, prohibiting their deployments on resource-limited devices.

**Present Work.** The extensive memory of KGNNs stems from the fact that all activations (*a.k.a.*, feature maps) in the forward pass need to be stored for gradient computation in the back propagation. Thus, training an $L$-layer KGNNs requires to cache all $L$ layers' intermediate activations, which dominates the GPU memory.

Inspired by recent activation compressed training techniques [2, 4, 9, 17, 26], we present TinyKG, a memory-efficient GPU-based training framework for KGNNs for the tasks of recommendation. In particular, TinyKG uses full-precision activations (*e.g.*, 32-bit floating point (FP32)) during the forward pass while storing a quantized version of activations (*e.g.*, 2-bit integer (INT2)) in the GPU buffers. During the backward pass, these low-precision activations are dequantized back to full-precision tensors to compute gradients. As such, our TinyKG largely reduces the memory footprint during training, allowing a larger batch size to fully utilize the power of neural message-passing mechanisms.

However, the quantization procedure introduces additional bias and variance, *i.e.*, the gap between the full-precision values and their quantized values, which inevitably affect the convergence and accuracy of KGNNs. To reduce the quantization error, TinyKG applies a uniform quantization with a stochastic rounding algorithm to compress the activations. We further show that our quantized strategy ensures unbiasedness with low variance. Therefore, the performance of quantized KGNNs is comparable to their original backbones without huge accuracy loss. We conduct extensive experiments to evaluate the effectiveness of the proposed TinyKG.

Our major contributions are summarized as follows:

- We propose a memory-saving training framework for KGNNs, namely TinyKG, which supports low-bit activation maps in the backward pass. As such, TinyKG can be easily adapted to many KGNN-based projects.
- We introduce a quantization strategy to efficiently compress the activations for back propagation. Besides, we show that our quantization algorithm is unbiased with well bounded variance, which performs well with small time overhead.
- We systematically analyze the TinyKG in terms of memory saving, time overhead, and accuracy loss on three public datasets. The results demonstrate that TinyKG can largely reduce memory footprint during training, with negligible loss in accuracy. Generally, TinyKG with INT2 quantization reduces the memory footprint of activation maps with 7×, and only with 2% loss in accuracy.

## 2 RELATED WORK

Our work is related to two lines of research: Knowledge-aware Recommendation and Scalable Graph Training. In this section, we briefly go through several prior efforts and discuss their limitations.

### 2.1 Knowledge-aware Recommendation

Knowledge-aware recommender systems have been successfully utilized to provide complementary information to alleviate the data sparsity or cold start issues [40, 41, 48]. The core idea is to transform the entities and relations of KGs into a compact embedding space, where one can impute the missing links/relations in the KGs, such as user-item links. Compared with triple-based models [1, 3, 5, 45, 48], such as TransE [1] and DistMult [45], KGNNs provide a class of more powerful architectures that are efficient for capturing multi-hop dependencies over the entire KGs [3, 40–42, 44]. Basically, KGNNs follow a recursive aggregation mechanism where each entity aggregates information from its immediate neighbors repeatedly, resulting in better performance.

There are several popular KGNNs in recommender systems, including R-GCN [34], KGCN [40], KGAT [41], CKAN [44], and KGIN [42]. For example, R-GCN [34] is the first study to show that the GNN framework can be applied for modeling multi-relational data in the KGs. CKAN [44] employs a heterogeneous propagation strategy to encode both collaborative user-item signals and knowledge-aware signals, and utilizes attention mechanisms to discriminate the contributions of different neighbors. Recently, KGIN [42] considers both user-item relationships at finer granularity of intents and long-range semantics of relational paths under the message-passing paradigm, which further improves the accuracy and explainability for recommendation.

Although significant research has focused on developing different architectures of KGNNs, reducing the training memory footprint of KGNNs is much less studied. As the message-passing schema in KGNNs needs to propagate entities' embeddings using multi-hop neighbors, the training of KGNNs is often slow and requires lots of memory. This poses a challenge when seeking to deploy KGNNs for industry-scale graphs [33, 39].

In this work, we focus on developing a general memory-saving training framework for KGNNs, by observing that most of the memory usage is mainly dominated by the activation maps during the back propagation. Therefore, we propose a new implementation for back propagation by compressing the activation maps, which significantly reduces the GPU memory usage. Interestingly, our memory-saving training framework can be seamlessly applied to many KGNNs since our TinyKG does not need to change the architectures of existing KGNNs.

## 2.2 Scalable Graph Training

The size of modern KGs has grown exponentially, *i.e.*, Microsoft Academic Graph has over 8 billion triples containing information about scientific publications and entities of related entity types [15]. Scaling up KG models is thus a critical need for massive KGs in the industry. There are many distributed frameworks for single-hop KG completion, where triples can be partitioned easily [24, 49, 50]. However, these frameworks cannot be directly used for training KGNNs due to complex multi-hop dependencies in the message-passing schema, especially when KGNNs go deep [16, 32]. Recently, SMORE [33], the first distributed framework for multi-hop KG models, is built upon a shared memory environment with multiple GPUs, while storing embedding parameters in the CPU memory to overcome the limited GPU memory. Our proposed TinyKG is orthogonal to SMORE and can be used to additionally reduce the activation maps of SMORE in the back propagation. We leave the potential combination as our future study.

On the other hand, great efforts have been made to train deep neural networks in resource-constraint scenarios [9, 10, 17, 21, 27, 29, 30, 46]. For example, model compression [21] and gradient compression [25] are able to reduce the storage and communication overhead by compressing the weights and gradients. Gradient checkpointing [10] trades computation for memory by dropping some of activations in the forward pass, and recomputing them in the backward pass. Swapping [22, 28] fully utilizes a large amount of host CPU memory by exchanging tensors between CPU and GPU. Nevertheless, swapping techniques do not fit memory-constrained devices as there is no sufficient host memory. Quantization-aware Training [13, 23, 38] generally aims to improve the model efficiency at the training or inference stage. However, many existing frameworks use full-precision data type to simulate the effect of real quantization (*a.k.a.* fake quantization) since many CUDA kernels cannot directly support low-bit operators/tensors. Also, the convergence behavior of quantization-aware training is still not well understood, limiting their generality.

In contrast, Activation Compressed Training [4, 9, 14, 17, 26] becomes a promising technique to reduce the training memory footprint as it only considers *storage*, allowing more flexible quantization strategies for better compression. This technique has been successfully used to train ResNet with 2-bit activations, reducing memory footprint by 12×, and enabling a 14× larger training batch size in practice [9, 17]. However, there is no existing work that extends this direction to KGNNs and analyzes its feasibility.

Our TinyKG is built upon this line of work, and provides a memory-efficient GPU implementation to support common graph convolutional operators in the KGNNs.

# 3 THE PROPOSED TINYKG

## 3.1 Problem Setup

We begin by describing the problem of knowledge-aware recommendation and introducing the notations.

**User-Item Graph.** In this work, we focus on learning user preferences from the implicit feedback (*e.g.*, click, comment, purchase, etc.). To be specific, we have a set of users $\mathcal{U} = \{u\}$ and a set of items $\mathcal{V} = \{v\}$. Let $\mathbf{Y}$ be the user-item interaction matrix, where $y_{uv} = 1$ indicates that user $u$ has engaged with item $v$ before, and $y_{uv} = 0$ otherwise.

**Knowledge Graph.** A KG is a multi-relational graph where nodes denote entities, and edges correspond to relations between entities. An edge in a KG represents a fact stored in the form of (*subject*, *predicate*, *object*). In recommendation scenarios, A KG often contains the side information, such as item attributes, taxonomy, or external knowledge. Formally, a KG is represented as $\mathcal{G} = \{(h, r, t)|h, t \in \mathcal{E}, r \in \mathcal{R}\}$, in which each triple $(h, r, t)$ indicates that a relation $r$ exists from head entity $h$ to tail entity $t$, $\mathcal{E}$ and $\mathcal{R}$ are the set of entities and relations in the KG, respectively. For example, the triple (*Tom Hanks*, *ActorOf*, *Forrest Gump*) states the fact that Tom Hanks is an actor of the movie Forrest Gump.

In knowledge-aware recommendation, the user-item graph $\mathbf{Y}$ can be seamlessly integrated with the knowledge graph $\mathcal{G}$ based on the item-entity alignment. An item $v \in \mathcal{V}$ corresponds to an entity $e \in \mathcal{E}$ (*e.g.*, the item "Forrest Gump" also appears in the KG as an entity). The set of entities $\mathcal{E}$ consists of items and item attributes. Normally, the KG provides complementary information for user-item graph, which highly alleviates the data sparsity or cold start issues.

**Task Description.** Given the user-item interaction graph $\mathbf{Y}$ and the knowledge graph $\mathcal{G}$, the task of knowledge-aware recommendation is to predict how likely a user would adopt an item that he/she has not interacted before. That is:

$$\hat{y}_{uv} = \text{KG-Model}(u, v|\mathbf{A}^{\mathcal{R}}, \mathbf{\Theta}),$$

where $\hat{y}_{uv}$ denotes the probability that user $u$ will engage with item $v$, $\mathbf{A}^{\mathcal{R}}$ represents the multi-relational matrix that can be constructed from $\mathbf{Y}$ and $\mathcal{G}$, and $\mathbf{\Theta}$ is the model parameters of the KG-Model.

## 3.2 Knowledge Graph Neural Networks

Knowledge Graph Neural Networks (KGNNs) have been shown great potential in improving diversity, accuracy, and explainability for personalized recommendation [40–42]. One of the key benefits of KGNNs is their ability of capturing high-order structural proximity among entities over the KGs, which alleviates the sparsity issue in recommendation.

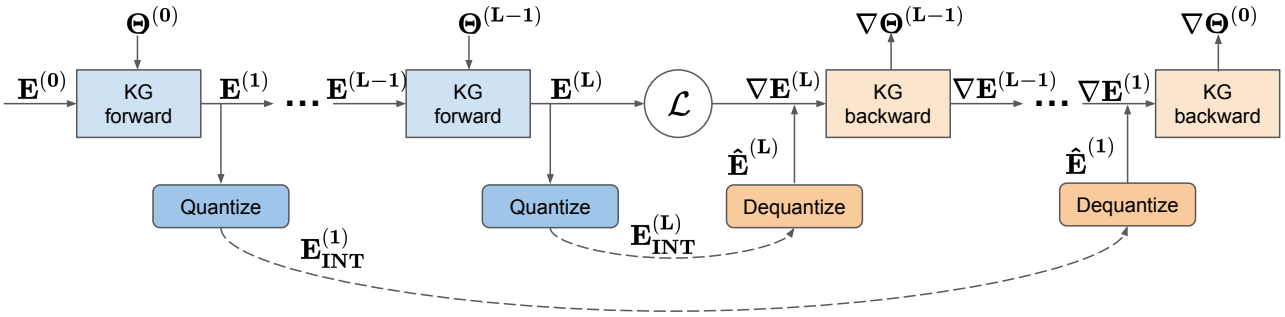


**Figure 1: The pipeline of the proposed TinyKG, where we quantize the full-precision activation maps $\{E^{(1)}, \cdots, E^{(L)}\}$ into lower numerical precision values $\{E^{(1)}_{INT}, \cdots, E^{(L)}_{INT}\}$ while still propagating the exact $\{E^{(1)}, \cdots, E^{(L)}\}$ during the forward pass. The low-precision activations are then dequantized back to full-precision tensor $\{\hat{E}^{(1)}, \cdots, \hat{E}^{(L)}\}$, which are used to compute gradients during the backward pass. Note that TinyKG only caches the quantized activations in the GPU buffers to reduce memory footprint.**

**Message-passing Schema.** Most of KGNNs fit under the message-passing schema, where the representation $\mathbf{e}_v$ of each entity is updated iteratively in each layer by collecting messages from its neighbors in a KG. In particular, the $l$-th layer can be simplified as:

$$\mathbf{E}^{(l+1)} = \text{KG-Layer}\left(\mathbf{A}^{\mathcal{R}}, \mathbf{E}^{(l)}, \mathbf{\Theta}^{(l)}\right), \quad l = 0, 1, \cdots, L-1. \tag{1}$$

where $\mathbf{E}^{(l)} \in \mathbb{R}^{N \times d}$ denotes the $d$-dimensional embeddings of entities at the $l$-th layer, $N$ is the number of entities in the KGs, $L$ is the number of layers, $\mathbf{A}^{\mathcal{R}}$ denotes the relational matrix that contains multi-type relationships among entities, $\mathbf{\Theta}^{(l)}$ denotes the trainable parameters in the $l$-th layer, and KG-Layer$(\cdot)$ is the propagation layer, such as graph convolutional layer [40], graph attention layer [41], and path-aware propagation layer [42]. After $L$ layers, a readout layer may be adopted to generate the final embedding for each entity. Finally, we can use downstream loss with different regularizations to optimize the model parameters [40–42].

Taking KGNN-LS [40] as an example, its aggregation process is: $\mathbf{E}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{\Theta}^{(l)})$, where $\hat{\mathbf{A}}$ is the normalized adjacency matrix of $\mathbf{A}^{\mathcal{R}}$ with self connection, and $\sigma(\cdot)$ is the non-linear function. Then, its computational graph can be decomposed as:

$$\begin{aligned}
\textit{Forward}: \quad \mathbf{H}^{(l)} &= \text{spmm}(\hat{\mathbf{A}}, \mathbf{E}^{(l)}) \ \downarrow \\
\mathbf{J}^{(l)} &= \text{mm}(\mathbf{H}^{(l)}, \mathbf{\Theta}^{(l)}) \ \downarrow \\
\mathbf{E}^{(l+1)} &= \sigma(\mathbf{J}^{(l)}), \\
\textit{Backward}: \quad \nabla_{\mathbf{J}^{(l)}} &= \text{ctx}(\mathbf{J}^{(l)}, \nabla_{\mathbf{E}^{(l+1)}}) \ \downarrow \\
(\nabla_{\mathbf{H}^{(l)}}, \nabla_{\mathbf{\Theta}^{(l)}}) &= \text{ctx}(\mathbf{H}^{(l)}, \mathbf{\Theta}^{(l)}, \nabla_{\mathbf{J}^{(l)}}) \ \downarrow \\
\nabla_{\mathbf{E}^{(l)}} &= \text{ctx}(\hat{\mathbf{A}}, \nabla_{\mathbf{H}^{(l)}}),
\end{aligned} \tag{2}$$

where spmm$(\cdot)$ is the sparse-dense matrix multiplication, mm$(\cdot)$ is the dense-dense matrix multiplication, $\nabla_{(\cdot)}$ denotes the gradient of activation/parameter that is always taken with respect to the loss $\mathcal{L}$, and ctx$(\cdot)$ denotes the context information that needs to be stored in the GPU memory for the backward pass, *i.e.*, $\nabla_{\mathbf{\Theta}^{(l)}} = \text{ctx}(\mathbf{H}^{(l)}, \nabla_{\mathbf{J}^{(l)}}) = \mathbf{H}^{(l)\top}\nabla_{\mathbf{J}^{(l)}}$. Essentially, the backward pass requires more context information than the forward pass.

**Memory Analysis.** In the inference stage, one can only perform the forward pass of the entire network, the results of the intermediate layers (*e.g.*, $\mathbf{H}^{(l)}$) can be discarded subsequently. However, in the training stage, the automatic differentiation engines (*e.g.*, PyTorch) need to store the following forward-pass variables in the memory to compute the gradients in the backward pass:

- The normalized adjacency matrix $\hat{\mathbf{A}}$: the matrix $\hat{\mathbf{A}}$ is often very sparse in KGs and only need to be kept once for all $L$ KG-Layers. Thus, the memory footprint of $\hat{\mathbf{A}}$ is trivial in the backward pass with space complexity $O(|\mathcal{E}|)$, where $|\mathcal{E}|$ denotes the number of edges in the KGs.
- The parameter of $\mathbf{\Theta}^{(l)} \in \mathbb{R}^{d \times d}$: the space complexity of $\mathbf{\Theta}^{(l)}$ is independent to the size of KGs (*e.g.*, $d \ll N$). The model parameters is generally negligible in recommendation with space complexity $O(Ld^2)$.
- All intermediate results $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times d}$, $\mathbf{J}^{(l)} \in \mathbb{R}^{N \times d}$, $\mathbf{E}^{(l)} \in \mathbb{R}^{N \times d}$ (if layer-aggregation is adopted [41, 42]): training an $L$-layer KGNN requires to cache all $L$ layers' intermediate outputs with a $O(LNd)$ space complexity. These intermediate outputs are termed as "activation maps" [2, 14, 17].

From the above analysis, it is the activation maps ($O(LNd)$) that mainly dominate the GPU memory during training, not the model parameters ($O(Ld^2)$). Based on this observation, in this work, we aim to reduce the capacity of intermediate activations by using principled quantized techniques to expedite the KGNNs' training.

## 3.3 Quantized Activation Maps

To reduce the memory consumption at the training stage, we present TinyKG, a memory-saving training framework for KGNNs without modifying their original architectures. Instead of saving the full-precision tensors (*e.g.*, FP32), TinyKG aims to lazily save quantized activation maps with lower numerical precision (*e.g.*, INT2) in the GPU buffers for back propagation.

Figure 1 is an overview of our proposed TinyKG. To be specific, the full-precision activations (*e.g.*, $\mathbf{E}^{(l)}$ ) is used during the forward pass. Then, the full-precision activations will be compressed into low-precision tensors via quantization (*e.g.*, $\mathbf{E}^{(l)}_{\text{INT}}$), which overwrites

the exact activations in the GPU buffers. The full-precision activations (*e.g.*, $\mathbf{E}^{(l)}$ ) can be then discarded in a layer-by-layer order. During the backward pass, TinyKG dequantizes the compressed activations in the GPU buffers back to full-precision tensors (*e.g.*, $\hat{\mathbf{E}}^{(l)}$). The gradients are then computed based on the dequantized activations. Therefore, the memory required for saving activation maps is highly reduced, enabling training with a larger batch size, or scaling up the model size, on the same GPU device.

In principle, any compression algorithm, either lossy or lossless, can be used to compress activation maps [36]. In this work, we introduce a simple yet effective quantized strategy to compress activation maps. Besides, we show that our quantized strategy is unbiased with low variance.

**Quantization (Float → Integer).** Specifically, the activation $\mathbf{E}^{(l)}$ (same for other activations) will be quantized and stored using $b$-bit integers. Let $B = 2^b - 1$ be the number of quantization bins, we quantize each tensor $\mathbf{e}_v^{(l)}$ of $\mathbf{E}^{(l)}$ as:

$$\mathbf{e}_{v_{INT}}^{(l)} = \text{Quant}(\mathbf{e}_v^{(l)}) = \left\lfloor \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B \right\rceil, \tag{3}$$

where $R_v^{(l)} = \max\{\mathbf{e}_v^{(l)}\} - \min\{\mathbf{e}_v^{(l)}\}$ is the range for $\mathbf{e}_v^{(l)}$, $Z_v^{(l)} = \min\{\mathbf{e}_v^{(l)}\}$ is the offset, $\mathbf{e}_{v_{INT}}^{(l)}$ is the compressed activation scaled to $[0, B]$, and $\lfloor \cdot \rceil$ denotes the stochastic rounding operator [12, 20]. For any scalar $x$, the stochastic rounding can be formulated as:

$$\lfloor x \rceil = \begin{cases} \lceil x \rceil, & \text{with probability } x - \lfloor x \rfloor, \\ \lfloor x \rfloor, & \text{with probability } 1 - (x - \lfloor x \rfloor), \end{cases}$$

where $\lceil \cdot \rceil$ is the ceil operator and $\lfloor \cdot \rfloor$ is the floor operator.

**Dequantization (Integer → Float).** During the backward pass, the compressed activation $\mathbf{e}_{v_{INT}}^{(l)}$ is dequantized as:

$$\hat{\mathbf{e}}_v^{(l)} = \text{Dequant}(\mathbf{e}_{v_{INT}}^{(l)}) = \frac{R_v^{(l)} \mathbf{e}_{v_{INT}}^{(l)}}{B} + Z_v^{(l)}, \tag{4}$$

where $\hat{\mathbf{e}}_v^{(l)}$ is a full-precision tensor that are then used to calculate the gradients for back propagation. All operators (*e.g.*, spmm($\cdot$)) are performed in full-precision. Note that the dequantized step is needed since most of the GPUs do not support low-bit operators, rather than full-precision and half-precision operators[1].

**Bias and Variance.** Inspired by recent efforts [9, 11, 26, 27], we have the following key property of the quantization:

Proposition 1. *The quantization of Eq. (3) and Eq. (4) for activation $\mathbf{e}_v^{(l)} \in \mathbb{R}^d$ is unbiased with well bounded variance, and its expectation and variance are:*

$$\mathbb{E}[\hat{\mathbf{e}}_v^{(l)}] = \mathbb{E}[\text{Dequant}(\text{Quant}(\mathbf{e}_v^{(l)}))] = \mathbf{e}_v^{(l)}, \qquad \text{Var}[\hat{\mathbf{e}}_v^{(l)}] \leq \frac{d[R_v^{(l)}]^2}{4B^2}. \tag{5}$$

The detailed analysis of Proposition 1 is given in Appendix. As the Dequant(Quant($\cdot$)) is an unbiased quantizer, the computed gradients are also unbiased. Nevertheless, the quantization inevitably imposes additional variance to the gradients during the backward pass, which plays an important role to the convergence behavior. As can be seen, the variance is inversely correlated with number of quantization bins $B$, *i.e.*, a larger $B$ leads to smaller variance. In the experiments, we will investigate how different values of $B$ affect the model performance (Sec 4.2.3).

To make our quantized strategy more rigorous, we theoretically show how much extra variance activation compression introduces. Let $\{\nabla_{\mathbf{\Theta}^{(l)}}, \nabla_{\mathbf{E}^{(l)}}\}$ be the full-precision gradients of $\{\mathbf{\Theta}^{(l)}, \mathbf{E}^{(l)}\}$, and $\{\hat{\nabla}_{\mathbf{\Theta}^{(l)}}, \hat{\nabla}_{\mathbf{E}^{(l)}}\}$ be the corresponding gradients using the compressed context. Further, we use the notation $\mathbf{G}_{\mathbf{\Theta}}^{(l\sim m)}(\hat{\nabla}_{\mathbf{E}^{(m)}}, \hat{\mathbf{C}}^{(m)})$ to represent the variance introduced by using the compressed context $\hat{\mathbf{C}}^{(m)}$. From Theorem 3 in [9], given an $L$-layer KGNN, we have:

$$\text{Var}\left[\hat{\nabla}_{\mathbf{\Theta}^{(l)}}\right] = \text{Var}\left[\nabla_{\mathbf{\Theta}^{(l)}}\right] + \sum_{m=l}^{L} \mathbb{E}\left[\text{Var}\left[\mathbf{G}_{\mathbf{\Theta}}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{E}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right) \mid \hat{\nabla}_{\mathbf{E}^{(m)}}\right]\right], \tag{6}$$

where $\text{Var}\left[\cdot|\hat{\nabla}_{\mathbf{E}^{(m)}}\right]$ is the conditional variance, and the $\text{Var}\left[\nabla_{\mathbf{\Theta}^{(l)}}\right]$ is the full-precision gradient variance in the original Stochastic Gradient Descent. Intuitively, the variance introduced by compressed context at different layers will accumulate as KGNNs go deep. Fortunately, most of KGNNs often have shallow architectures (*e.g.*, $L \leq 4$) [40–42], the gradient variance is thus trivial comparing to deep CNNs in practice [9].

According to Eq. (5) and Eq. (6), we may reduce the numerical precision for free, as the quantization is unbiased and the variance is negligible. This suggests that there is no need to adopt expensive sophisticated quantization strategies, like mixed-precision quantization or non-uniform quantization, as considered in previous work [9, 35]. Moreover, it is worth noting that our proposed TinyKG is easily compatible to any KGNNs since it only changes the routine of storage saving in the GPU buffers, rather than their vanilla network architectures.

# 4 EXPERIMENTS

In real-world applications, the deployed models should achieve a balanced trade-off among model performance, speed, and space complexity. In this section, we systematically analyze the proposed TinyKG in terms of memory saving, time overhead, and accuracy loss on three real-world datasets.

## 4.1 Experimental Settings

*4.1.1* ***Datasets.*** We conduct experiments on three publicly available benchmark datasets: Amazon-book, MovieLens-20M, and Yelp, which vary in terms of domain, size, and sparsity:

- **Amazon-book[2]:** The dataset contains a large corpus of user reviews, ratings, and product metadata, collected from the Amazon Book category. To guarantee the quality of the dataset, we use the 10-core setting, retaining users and items with at least ten interactions.
- **MovieLens-20M[3]:** The dataset is a widely used benchmark dataset in recommendations, which consists of approximately 20 million explicit ratings (ranging from 1 to 5). We transform

[1] https://pytorch.org/blog/accelerating-training-on-nvidia-gpus-with-pytorch-automatic-mixed-precision/.

[2] http://jmcauley.ucsd.edu/data/amazon.

[3] https://grouplens.org/datasets/movielens/20m/.

**Table 1: The statistics of the benchmark datasets.**

| | | Amazon-book | MovieLens-20M | Yelp2018 |
|---|---|---|---|---|
| User-Item Graph | #Users | 70, 679 | 138, 159 | 45, 919 |
| | #Items | 24, 915 | 16, 954 | 45, 538 |
| | #Interactions | 847, 733 | 13, 501, 622 | 1, 185, 068 |
| Knowledge Graph | #Entities | 88, 572 | 102, 569 | 90, 961 |
| | #Relations | 39 | 32 | 42 |
| | #Triples | 2, 557, 746 | 499, 474 | 1, 853, 704 |

them into implicit feedback, where each user-item interaction is marked with 1 if its rating score is greater than 3, otherwise 0.

- **Yelp**[4]**:** This dataset is adopted from the 2018 edition of the Yelp challenge. In this work, we view the local businesses like restaurants and bars as items. Similarly, we use the 10-core setting to ensure that each user and item have at least ten interactions.

In addition to the user-item graph, we directly follow the previous work [40–42, 44] to construct an item knowledge graph for each dataset. Then, we merge the user-item graph and the item knowledge graph via item alignments. In particular, we consider triples from the item knowledge graph that are directly related to the entities to align with the items in the user-item graph, regardless of which role it plays (*i.e.*, subject or object).

Table 1 briefly summarizes the statistics of those three datasets. For each dataset, we randomly select 80% of interaction history of each user to constitute the training set, and treat the remaining as the test set. From the training set, we randomly select 10% of interactions as the validation set to tune hyper-parameters [41, 42].

*4.1.2* ***Baselines.*** The main goal of our TinyKG is to reduce the training GPU memory of the KGNNs, not to design new architectures. Therefore, we evaluate our framework on existing state-of-the-art KGNNs, including:

- **KGAT** [41]: KGAT utilizes an attentive neighborhood aggregation mechanism on the KG to generate user/item representations. As such, it is able to discriminate the important of neighbors during propagation, leading to more accurate and explainable recommendation.
- **KGNN** [40]: KGNN first transforms the heterogeneous KG into a user-specific weighted graph, and then computes the personalized item embedding based on graph neural network. It also introduces the label smoothness regularization to provide better inductive bias.
- **KGIN** [42]: KGIN is the latest state-of-the-art propagation-based recommendation method, which explicitly models user interaction behaviors with latent intents. It adopts a relational path-aware aggregation schema to capture the long-range dependencies in the KG.

The baselines are using full-precision training, while the proposed TinyKG will compress their corresponding activation maps in the GPU buffers during training. However, their behaviors are identical at the inference stage.

*4.1.3* ***Evaluation Metrics.*** We follow the previous work [41, 42] to conduct the evaluation of top-$K$ recommendation. For each user in the test set, we treat all the items that the user has not interacted with as the negative items. Then each KGNN model predicts the user's preference scores over all the items, except the positive ones in the training set. To evaluate the effectiveness of different approaches, we adopt two widely-used top-$K$ evaluation protocols [41, 42]: Recall@$K$ and NDCG@$K$. By default, we set $K = 20$, and we report the average metrics for all users in the test set.

*4.1.4* ***Implementation Details.*** We implement all KGNNs in PyTorch, and run the experiments on a Linux machine with a single NVIDIA Tesla P100 with 16 GB GPU memory. For a fair comparison, we fix the embedding size of the entities (*e.g.*, Eq. (1)) as 64, the training batch size as 1024, the number of KGNN layer as 3, the learning rate as $1e^{-3}$, and the optimizer as Adam for all the baselines. For other model-specific hyper-parameters, we use the default settings as suggested in the original papers.

For TinyKG, the common operators (*e.g.,* Linear, ReLU, BatchNorm, SPMM, SPMM_MAX, etc.) are built on the top of existing frameworks [9, 26, 27, 31] with different configurations using CUDA kernels. All the substitutions within KGNNs can be done automatically with a model converter. Taking ReLU$(\cdot)$ as an example, we have $\mathbf{y} = \text{ReLU}(\mathbf{x}) = \mathbf{x} \odot \mathbf{1}_{\mathbf{x}>0}$ and $\nabla_{\mathbf{y}} = \nabla_{\mathbf{y}} \odot \mathbf{1}_{\mathbf{x}>0}$. Here, ReLU$(\cdot)$ only needs to store $\mathbf{1}_{\mathbf{x}>0}$ for the backward pass, which takes one bit per element in the buffers. As PyTorch only supports precision down to INT8, our TinyKG can further convert quantized tensors into bit-streams by CUDA kernels to maximize the memory saving. In addition, our TinyKG is able to support both full-precision and half-precision (*e.g.*, bfloat16) that is thus compatible with Automated Mixed Precision training (AMP)[5], to further reduce the memory consumption.

## 4.2 Main Results

*4.2.1* ***Overall Performance.*** The performance of our proposed TinyKG depends on the number of quantization bins $B$. To evaluate how the quantized level affects the model performance, we vary the $B$ within $[8, 4, 2, 1]$. Ideally, a larger $B$ can achieve the closer behavior as full-precision settings, but with more memory footprint. Table 2, Table 3, and Table 4 show the performance of KGNNs on Amazon-book, MovieLens-20M, and Yelp2018, respectively.

[4]https://www.yelp.com/dataset.

[5]https://pytorch.org/docs/stable/amp.html

**Table 2: The performance of KGNNs trained on the Amazon-book dataset with compressed activations storing in different precision. All results are averaged over ten random trials. All number are percentage numbers without %.**

| Model | Metric | FP32(baseline) | INT8 | INT4 | INT2 | INT1 |
|---|---|---|---|---|---|---|
| KGAT | Recall@20 (%) | 14.85±0.15 | 14.81±0.12 | 14.78±0.14 | 14.66±0.14 | 14.08±0.15 |
| | NDCG@20 (%) | 8.12±0.10 | 8.10±0.12 | 8.08±0.11 | 7.98±0.12 | 7.65±0.17 |
| KGNN | Recall@20 (%) | 13.64± 0.16 | 13.61± 0.18 | 13.58±0.12 | 13.46±0.12 | 12.91±0.13 |
| | NDCG@20 (%) | 5.71±0.09 | 5.70± 0.07 | 5.67±0.10 | 5.62±0.09 | 5.39±0.11 |
| KGIN | Recall@20 (%) | 16.89±0.16 | 16.85± 0.13 | 16.82±0.12 | 16.65±0.12 | 15.97±0.11 |
| | NDCG@20 (%) | 9.16±0.12 | 9.14± 0.14 | 9.10±0.12 | 8.99±0.15 | 8.63±0.12 |

**Table 3: The performance of KGNNs trained on the MovieLens-20M dataset with compressed activations storing in different precision. All results are averaged over ten random trials. All number are percentage numbers without %.**

| Model | Metric | FP32(baseline) | INT8 | INT4 | INT2 | INT1 |
|---|---|---|---|---|---|---|
| KGAT | Recall@20 (%) | 22.13± 0.13 | 22.09± 0.11 | 21.96±0.13 | 21.73±0.14 | 20.85±0.17 |
| | NDCG@20 (%) | 15.73± 0.07 | 15.70± 0.10 | 15.61±0.11 | 15.45±0.12 | 14.98±0.14 |
| KGNN | Recall@20 (%) | 21.04± 0.11 | 20.99± 0.13 | 20.85±0.13 | 20.63±0.11 | 20.01±0.12 |
| | NDCG@20 (%) | 13.26± 0.10 | 13.24± 0.08 | 13.15±0.10 | 13.02±0.10 | 12.50±0.11 |
| KGIN | Recall@20 (%) | 24.58± 0.11 | 24.54± 0.12 | 24.39±0.13 | 24.14±0.13 | 23.65±0.14 |
| | NDCG@20 (%) | 18.05± 0.14 | 18.01± 0.15 | 17.92±0.14 | 17.75±0.15 | 17.21±0.12 |

**Table 4: The performance of KGNNs trained on the Yelp2018 dataset with compressed activations storing in different precision. All results are averaged over ten random trials. All number are percentage numbers without %.**

| Model | Metric | FP32(baseline) | INT8 | INT4 | INT2 | INT1 |
|---|---|---|---|---|---|---|
| KGAT | Recall@20 (%) | 7.14± 0.04 | 7.12± 0.01 | 7.11±0.03 | 7.02±0.02 | 6.73±0.04 |
| | NDCG@20 (%) | 8.78± 0.02 | 8.76± 0.03 | 8.73±0.02 | 8.63±0.02 | 8.28±0.03 |
| KGNN | Recall@20 (%) | 6.83± 0.02 | 6.81± 0.03 | 6.80±0.02 | 6.70±0.03 | 6.43±0.03 |
| | NDCG@20 (%) | 7.87± 0.03 | 7.86± 0.03 | 7.83±0.02 | 7.74±0.01 | 7.43±0.02 |
| KGIN | Recall@20 (%) | 7.79± 0.03 | 7.78± 0.02 | 7.74±0.01 | 7.67±0.02 | 7.35±0.02 |
| | NDCG@20 (%) | 9.01± 0.02 | 8.99± 0.01 | 8.96±0.02 | 8.84±0.03 | 8.54±0.03 |

From the tables, we can observe that our TinyKG can consistently achieve comparable performance as baselines. For example, the loss in accuracy is less than 0.3% when using the INT8 quantization. And the INT2 quantization only causes a small accuracy drop ($< 2\%$) for all cases. Even with the extreme INT1 quantization, the accuracy loss is still acceptable, usually $< 6\%$ in almost all experiments, in terms of both Recall@20 and NDCG@20. In contrast, adopting the INT1 or INT2 quantizations will cause a significant accuracy drop for CNNs (usually $> 6\%$) [9].

In addition, we visualize the training loss curves of KGNNs with/without TinyKG in the Figure 2. As can be seen, all curves of KGNNs under TinyKG with INT2 quantization are consistent with their original baselines. Therefore, the performance of proposed TinyKG can achieve quite comparable results with the state-of-the-art in terms of accuracy.

*4.2.2* ***Memory and Speed.*** As discussed before, our TinyKG is able to reduce the training memory footprint. However, we notice that TinKG slightly slows down the training speed. The extra time cost comes directly from the (de)quantization. Table 5 shows the trade-off among the memory, speed, and performance. Here we only show the results on Amazon-Book, and the observations are similar for other two datasets. We simply omit their results.

From Table 5, we note that lower precision quantization can save more memory footprint, with smaller time overhead, but with larger performance drop. INT1 quantization can aggressively reduces the memory footprint with 7.1×, yet with around 5% performance drop in many situations. In practice, one can safely adopt INT2 quantization for training KGNNs to achieve a good balance. For instance, KGAT with INT2 quantization generally reduces the memory footprint of activation maps with 7.1×, but merely with 1.26% loss in accuracy, while the time overhead is roughly 25%; KGIN with INT2 quantization reduces 7.58× the memory footprint of activations with just 1.43% performance drop, and the time overhead roughly ranges from 17%to 25%.

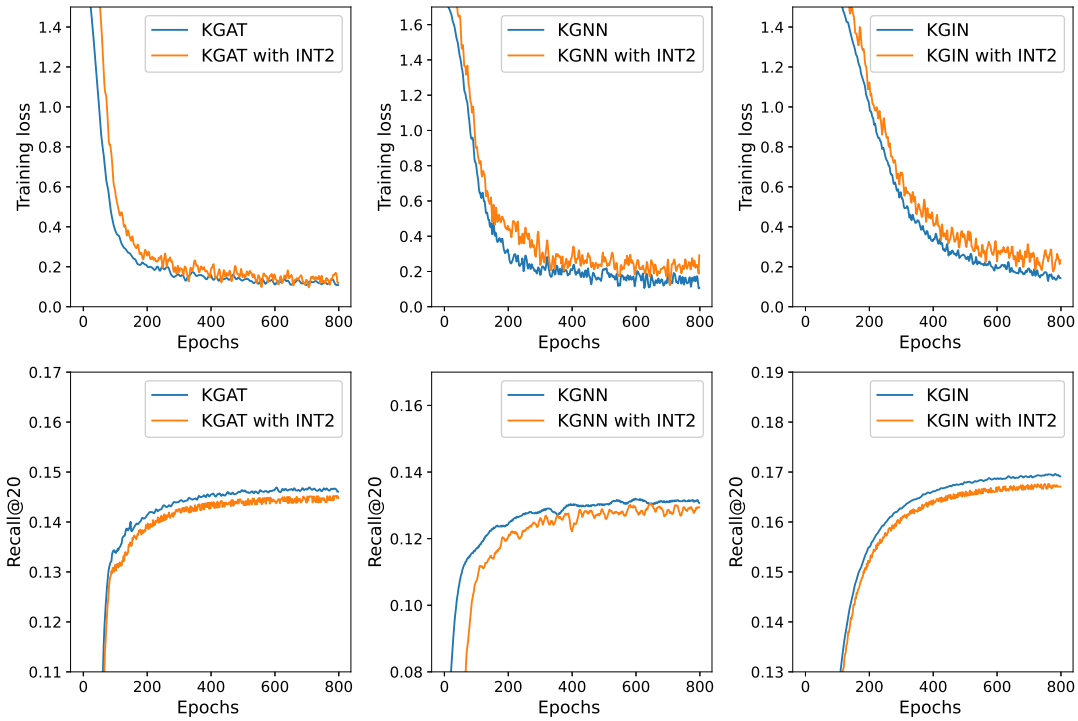


**Figure 2: Training curves comparison for KGNNs with/without TinyKG (INT2) for Amazon-Book.**

**Table 5: Comparison on the test accuracy, running time, and memory saving on Amazon-book dataset. "Act Mem" is the memory (MB) occupied by activation maps. "GPU Time" indicates the running time of one epoch. "Acc Loss" denotes the performance drop in term of Recall@20. All reported results are averaged over ten random trials, and the average results are reported.**

| | | FP32 | INT8 | INT4 | INT2 | INT1 |
|---|---|---|---|---|---|---|
| KGAT | Act Mem (MB) | 1117.1 | 503.5(2.22×) | 377.5(2.96×) | 157.3(7.10×) | 112.3(9.95×) |
| | GPU Time (Sec) | 321.9 | 498.1(+54.74%) | 432.6(+34.39%) | 402.4(+25.01%) | 386.2(+19.98%) |
| | Acc Loss | −0% | −0.26% | −0.47% | −1.26% | −5.01% |
| KGNN | Act Mem (MB) | 1358.6 | 845.3(1.61×) | 495.2(2.74×) | 187.9(7.23×) | 132.5(10.2×) |
| | GPU Time (Sec) | 303.2 | 467.3(+54.12%) | 413.2(+36.28%) | 379.0(+25.00%) | 337.2(+11.21%) |
| | Acc Loss | −0% | −0.21% | −0.41% | −1.31% | −5.30% |
| KGIN | Act Mem (MB) | 1274.3 | 674.0(1.89×) | 457.6(2.78×) | 168.2(7.58×) | 119.2(10.6×) |
| | GPU Time (Sec) | 378.8 | 524.2(+38.38%) | 471.3(+24.42%) | 445.0(+17.72%) | 410.4(+8.40%) |
| | Acc Loss | −0% | −0.23% | −0.44% | −1.43% | −5.42% |

As such, one can choose a larger batch size or a more complex network architecture to fully utilize the power of neural message-passing mechanisms with the relaxed memory footprint restriction [35]. That is, our TinyKG unveils the great value by providing more choices to explore the design of KGNNs.

*4.2.3* ***The effects of ratio*** $\frac{d}{B^2}$. From the property of our quantized strategy given in Proposition 1, the calculated gradients are unbiased. However, the quantization also imposes extra variance to the calculated gradients, where the variance linearly scales with the ratio $\frac{d}{B^2}$. Note that the $R_v^{(l)}$ is tuned on-the-fly. To quantitatively study the effect of the extra variance, we fix $B = 2$ (INT2 quantization), and vary $d$ within $\{32, 64, 96, 128\}$. We show the performance of KGNNs on three datasets.

From Figure 3, we observe that the loss in accuracy is generally negligible. This can be explained by the following: 1) the model depth of KGNNs is much smaller than CNNs due to the over-smoothing problem. The accumulate quantized errors are thus relatively small according to Eq. (6). 2) Although the quantized steps introduce extra noise in gradients during the backward pass, recent work [18] finds that simple noisy regularization can be an effective way to address over-smoothing for message-passing schema, while

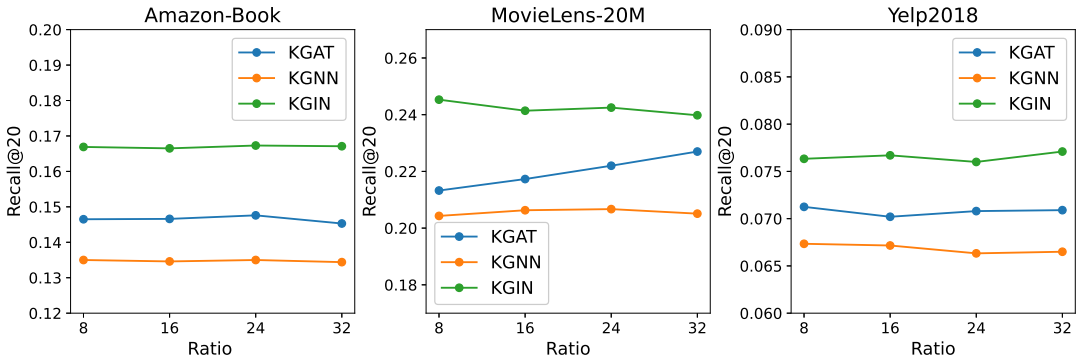


**Figure 3: The sensitivity study of TinyKG (INT2) to the ratio $\frac{d}{B^2}$ on three datasets.**

**Table 6: The performance (Recall@20) of KGNNs trained on the Amazon-book dataset under stochastic rounding (SR) and nearest rounding (NR). All results are averaged over ten random trials. All number are percentage numbers without %, and '-' means the algorithm fails to converge with > 40% performance drop**

| Model | Metric | FP32(baseline) | INT8 | INT4 | INT2 | INT1 |
|---|---|---|---|---|---|---|
| KGAT | SR(%) | 14.85± 0.15 | 14.81±0.12 | 14.78±0.14 | 14.66±0.14 | 14.08±0.15 |
| | NR(%) | 14.85±0.15 | 10.04±0.12 | - | - | - |
| KGNN | SR(%) | 13.64±0.16 | 13.61±0.18 | 13.58±0.12 | 13.46±0.12 | 12.91±0.13 |
| | NR(%) | 13.64±0.16 | 9.87±0.17 | - | - | - |
| KGIN | SR(%) | 16.89±0.16 | 16.85±0.13 | 16.82±0.12 | 16.65±0.12 | 15.97±0.11 |
| | NR(%) | 16.89±0.16 | 13.89±0.24 | - | - | - |

CNNs do not have such property [9]. Therefore, KGNNs are much more noise-tolerant, allowing for lower bit quantizations.

*4.2.4* ***Stochastic Rounding vs. Nearest Rounding.*** To explore the effect of stochastic rounding in TinyKG, we compare it with the commonly used nearest rounding to evaluate how different rounding algorithms affect the performance. Both stochastic rounding and nearest rounding share similar memory footprint during training since the memory cost is determined by the number of quantization bins. In this work, we mainly compare their performance on Amazon-Book in terms of Recall@20. As Table 6 shows, nearest rounding fails to converge at most cases. This suggests that stochastic rounding is important to guarantee good performance in TinyKG.

## 5 CONCLUSION AND FUTURE WORK

In this paper, we propose TinyKG, a simple-yet-effective framework for training KGNNs with compressed activation maps. Specifically, we leverage a uniform quantization with a stochastic rounding algorithm to efficiently compress the intermediate activations while training KGNNs. In addition, we verify the unbiasedness and low variance of our introduced quantization. To evaluate the performance of our method, we conduct comprehensive experiments over three real-world datasets for downstream recommendation tasks. In the experiments, we systematically analyze the trade-off of our strategy among the memory-saving, time overhead, and accuracy drop. The experimental results demonstrate that TinyKG can extensively reduce the GPU's memory footprint, while merely incurring a slight time overhead and performance drop. In addition, we show that our proposed TinyKG can be successfully applied to existing KGNNs without much extra engineering.

As our future work, we would like to design self-supervised KGNNs under our TinyKG. The graph contrastive learning framework [47] can learn local and global node representations to better capture structure information. Nevertheless, the contrastive frameworks usually require a large batch size of comparing pairs to obtain more accurate estimation of the contrastive loss, leading to large GPU memory. We plan to reduce the training memory footprint of constrative framework using our compressed technique. Moreover, our TinyKG is orthogonal to most of existing techniques, including distributed training [33], compression [21] and gradient compression [25]. We are interested to explore the potential benefits of integrating TinyKG with those efforts.

## APPENDIX

The quantization of Eq. (3) and Eq. (4) for activation $\mathbf{e}_v^{(l)} \in \mathbb{R}^d$ is unbiased, its expectation and variance are:

$$\mathbb{E}[\hat{\mathbf{e}}_v^{(l)}] = \mathbb{E}[\text{Dequant}(\text{Quant}(\mathbf{e}_v^{(l)}))] = \mathbf{e}_v^{(l)}, \quad \text{Var}[\hat{\mathbf{e}}_v^{(l)}] \leq \frac{d[R_v^{(l)}]^2}{4B^2}.$$

Proof. Based on the definition of stochastic rounding [12, 20] and the fact that $\lceil x \rceil = 1 + \lfloor x \rfloor$, we have:

$$\begin{aligned}\mathbb{E}[\lfloor x \rceil] &= \lceil x \rceil \cdot (x - \lfloor x \rfloor) + \lfloor x \rfloor \cdot (1 - (x - \lfloor x \rfloor)) \\ &= (1 + \lfloor x \rfloor) \cdot (x - \lfloor x \rfloor) + \lfloor x \rfloor \cdot (1 - x + \lfloor x \rfloor) \\ &= x\end{aligned}$$

Similar, we have:

$$\begin{aligned}\mathbb{E}[\hat{\mathbf{e}}_v^{(l)}] &= \mathbb{E}[\text{Dequant}(\text{Quant}(\mathbf{e}_v^{(l)}))] \\ &= \mathbb{E}\left[\frac{R_v^{(l)}}{B} \cdot \left\lfloor \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B \right\rceil + Z_v^{(l)}\right] \\ &= \frac{R_v^{(l)}}{B} \cdot \mathbb{E}\left[\left\lfloor \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B \right\rceil\right] + Z_v^{(l)} \\ &= \frac{R_v^{(l)}}{B} \cdot \left(\frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B\right) + Z_v^{(l)} \\ &= \mathbf{e}_v^{(l)}.\end{aligned}$$

For variance,

$$\begin{aligned}\text{Var}[\lfloor x \rceil] &= (\lceil x \rceil - x)^2 \cdot (x - \lfloor x \rfloor) + (\lfloor x \rfloor - x)^2 \cdot (1 - x + \lfloor x \rfloor) \\ &= (1 - (x - \lfloor x \rfloor))^2 \cdot (x - \lfloor x \rfloor) + (x - \lfloor x \rfloor)^2 \cdot (1 - (x - \lfloor x \rfloor)) \\ &= -(x - \lfloor x \rfloor)^2 + (x - \lfloor x \rfloor)\end{aligned}$$

As such, let $\bar{\mathbf{e}} = [\bar{e}_1, \cdots, \bar{e}_d] = \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B$, we have:

$$\begin{aligned}\text{Var}[\hat{\mathbf{e}}_v^{(l)}] =& \text{Var}\left[\frac{R_v^{(l)}}{B} \cdot \left\lfloor \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B \right\rceil + Z_v^{(l)}\right] \\ &= \frac{[R_v^{(l)}]^2}{B^2} \cdot \text{Var}\left[\cdot \left\lfloor \frac{\mathbf{e}_v^{(l)} - Z_v^{(l)}}{R_v^{(l)}} B \right\rceil\right] \\ &= \frac{[R_v^{(l)}]^2}{B^2} \cdot \sum_{i=1}^{d} \text{Var}[\lfloor \bar{e}_i \rceil] \\ &= \frac{[R_v^{(l)}]^2}{B^2} \cdot \sum_{i=1}^{d} [-(\bar{e}_i - \lfloor \bar{e}_i \rfloor)^2 + (\bar{e}_i - \lfloor \bar{e}_i \rfloor)] \\ &= \frac{[R_v^{(l)}]^2}{B^2} \cdot \sum_{i=1}^{d} [-(\bar{e}_i - \lfloor \bar{e}_i \rfloor - \frac{1}{2})^2 + \frac{1}{4}] \\ &\leq \frac{d[R_v^{(l)}]^2}{4B^2}.\end{aligned}$$

The inequality always holds since $\bar{e}_i - \lfloor \bar{e}_i \rfloor \in [0, 1]$. And the upper bound is achieved when $\bar{e}_i - \lfloor \bar{e}_i \rfloor = \frac{1}{2}$.

□

## REFERENCES


[1] 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).

[2] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In *Advances in Neural Information Processing Systems*.

[3] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *The world wide web conference*. 151–161.

[4] Ayan Chakrabarti and Benjamin Moseley. 2019. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems* 32 (2019).

[5] Huiyuan Chen and Jing Li. 2020. Learning data-driven drug-target-disease interaction via neural tensor network. In *International Joint Conference on Artificial Intelligence*.

[6] Huiyuan Chen, Yusan Lin, Fei Wang, and Hao Yang. 2021. Tops, bottoms, and shoes: building capsule wardrobes via cross-attention tensor network. In *Fifteenth ACM Conference on Recommender Systems*. 453–462.

[7] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2021. Structured graph convolutional networks with stochastic masks for recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 614–623.

[8] Huiyuan Chen, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2022. Graph Neural Transport Networks with Non-local Attentions for Recommender Systems. In *Proceedings of the ACM Web Conference 2022*. 1955–1964.

[9] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. 2021. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*. 1803–1813.

[10] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[11] Michael P Connolly, Nicholas J Higham, and Theo Mary. 2021. Stochastic rounding and its probabilistic backward error analysis. *SIAM Journal on Scientific Computing* (2021), A566–A585.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems* (2015).

[13] Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. 2021. VQ-GNN: A Universal Framework to Scale up Graph Neural Networks using Vector Quantization. *Advances in Neural Information Processing Systems* (2021).

[14] R David Evans and Tor Aamodt. 2021. AC-GC: Lossy Activation Compression with Guaranteed Convergence. *Advances in Neural Information Processing Systems* 34 (2021).

[15] Michael Färber. 2019. The microsoft academic knowledge graph: a linked data source with 8 billion triples of scholarly data. In *International semantic web conference*. Springer, 113–129.

[16] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*. 3294–3304.

[17] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *International Conference on Machine Learning*. 3304–3314.

[18] Jonathan Godwin, Michael Schaarschmidt, Alexander L Gaunt, Alvaro Sanchez-Gonzalez, Yulia Rubanova, Petar Veličković, James Kirkpatrick, and Peter Battaglia. 2022. Simple GNN Regularisation for 3D Molecular Property Prediction and Beyond. In *International Conference on Learning Representations*.

[19] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[20] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International conference on machine learning*. 1737–1746.

[21] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations* (2016).

[22] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1341–1355.

[23] Qing Jin, Jian Ren, Richard Zhuang, Sumant Hanumante, Zhengang Li, Zhiyu Chen, Yanzhi Wang, Kaiyuan Yang, and Sergey Tulyakov. 2022. F8Net: Fixed-Point 8-bit Only Multiplication for Network Quantization. In *International Conference on Learning Representations*.

[24] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems* 1 (2019), 120–131.

[25] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2018. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*.

[26] Xiaoxuan Liu, Lianmin Zheng, Dequan Wang, Yukuo Cen, Weize Chen, Xu Han, Jianfei Chen, Zhiyuan Liu, Jie Tang, Joey Gonzalez, et al. 2022. GACT: Activation Compressed Training for Generic Network Architectures. In *International Conference on Machine Learning*. 14139–14152.

[27] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. 2021. EXACT: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*.

[28] Chen Meng, Minmin Sun, Jun Yang, Minghui Qiu, and Yang Gu. 2017. Training deeper models by GPU memory optimization on TensorFlow. In *Proc. of ML Systems Workshop in NIPS*.

[29] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *International Conference on Learning Representations*.

[30] Hesham Mostafa and Xin Wang. 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*. 4646–4655.

[31] Zizheng Pan, Peng Chen, Haoyu He, Jing Liu, Jianfei Cai, and Bohan Zhuang. 2021. Mesa: A Memory-saving Training Framework for Transformers. *arXiv preprint arXiv:2111.11124* (2021).

[32] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Anand Sivasubramaniam, and Mahmut Kandemir. 2020. Gcn meets gpu: Decoupling "when to sample" from "how to sample". *Advances in Neural Information Processing Systems* (2020).

[33] Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. 2022. SMORE: Knowledge Graph Completion and Multi-hop Reasoning in Massive Knowledge Graphs. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*.

[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] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. In *International Conference on Learning Representations*.

[36] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2020. And the Bit Goes Down: Revisiting the Quantization of Neural Networks. In *International Conference on Learning Representations*.

[37] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. 2018. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 297–305.

[38] Shyam Anil Tailor, Javier Fernandez-Marques, and Nicholas Donald Lane. 2021. Degree-Quant: Quantization-Aware Training for Graph Neural Networks. In *International Conference on Learning Representations*.

[39] Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*. 9448–9457.

[40] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 968–977.

[41] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 950–958.

[42] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the Web Conference 2021*. 878–887.

[43] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. 2022. Improving Fairness in Graph Neural Networks via Mitigating Sensitive Attribute Leakage. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*.

[44] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. 2020. CKAN: collaborative knowledge-aware attentive network for recommender systems. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 219–228.

[45] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*.

[46] Chin-Chia Michael Yeh, Mengting Gu, Yan Zheng, Huiyuan Chen, Javid Ebrahimi, Zhongfang Zhuang, Junpeng Wang, Liang Wang, and Wei Zhang. 2022. Embedding Compression with Hashing for Efficient Representation Learning in Graph. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*.

[47] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.

[48] 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.

[49] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 739–748.

[50] Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. 2019. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*. 2494–2504.