Optimizing Memory Efficiency of Graph Neural Networks on Edge Computing Platforms

[Brief Industry Paper]

Ao Zhou[1,2], Jianlei Yang[2], Yeqi Gao[2], Tong Qiao[2], Yingjie Qi[2], Xiaoyi Wang[1], Yunli Chen[1],
Pengcheng Dai[3], Weisheng Zhao[4] and Chunming Hu[2]

[1]School of Software, Beijing University of Technology, Beijing, China
[2]School of Computer Science and Engineering, Beihang University, Beijing, China
[3]Beijing Bytedance Technology Co., Ltd, Beijing, China
[4]School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China

***Abstract*—Graph neural networks (GNN) have achieved state-of-the-art performance on various industrial tasks. However, the poor efficiency of GNN inference and frequent Out-Of-Memory (OOM) problem limit the successful application of GNN on edge computing platforms. To tackle these problems, a feature decomposition approach is proposed for memory efficiency optimization of GNN inference. The proposed approach could achieve outstanding optimization on various GNN models, covering a wide range of datasets, which speeds up the inference by up to 3×. Furthermore, the proposed feature decomposition could significantly reduce the peak memory usage (up to 5× in memory efficiency improvement) and mitigate OOM problems during GNN inference.**

I. Introduction

In recent years, as a generalization of conventional deep learning methods on the non-Euclidean domain, Graph Neural Networks (GNN) are widely applied to various fields of research, such as node classification [1], link prediction [2] and feature matching [3]. In this paper, we propose a memory efficient method - feature decomposition, for GNN inference on hardware resource-limited platforms (mobiles, edge devices, etc.). We focus on two major problems in GNN inference: 1) *Poor inference efficiency* - The irregular graph structure and large vertex feature length pose difficulties in efficient GNN inference on edge devices, and 2) *Frequent Out-Of-Memory (OOM) problem* - The feature vectors with high dimensionality occupy enormous space in memory that often exceeds the memory limit.

The Message Passing based GNN inference can be summarized as two distinct phases: Combination and Aggregation. The former updates feature vector of each vertex with MLP operations, while the latter updates the vectors by aggregating features in their neighborhoods. The majority of current works on GNN optimization focus on large-scale high performance systems with abundant resources (GPU/Memory/etc.) [4], [5]. Unfortunately, there exists little research explored for improving GNN inference efficiency on CPU-only edge devices. In this paper, we propose a novel approach to optimize GNN memory efficiency from a new perspective: decomposing the dimension of feature vectors and performing aggregation respectively. Considering that aggregate operation is an element-wise operation on feature vectors, decomposing the feature vectors of all vertices and performing aggregation separately will not harm the accuracy of GNN inference. With our method, the data reuse of neighbor feature vectors is improved by loading more feature data into cache, which greatly improves aggregation efficiency. Besides, the greatly reduced feature dimension in each aggregate operation also reduces the risk of encountering OOM.

To evaluate our feature decomposition method, we conduct sufficient experiments with PyG framework [6]. Our feature decomposition approach can be easily implemented based on Gather-ApplyEdge-Scatter (GAS) abstraction of PyG. The proposed approach is evaluated for different GNN inference models with various datasets and could obtain about 3× speedups compared with PyG baseline. Furthermore, our approach could significantly improve the memory efficiency by reducing the cache miss rate and peak memory usage.

II. Feature Decomposition

We first profile GNNs performance on CPU to figure out their workload characteristics, and then describe the proposed memory optimization approaches in detail.

A. Characterizing GNNs on CPU

To identify the computation bottleneck of GNNs on resource-limited CPU, we conduct quantitative characterizations using PyG on Intel CPU. Three classic GNN models are chosen as evaluation objects, including GCN [1], GraphSage (GSC) [7] and GAT [8].

Execution Time Breakdown. We evaluate GNNs inference on several datasets [7], [9], [10], and the percentage of aggregation runtime comparison is illustrated in Fig. 1. Due to the large scale of graph and long feature vectors, the aggregation phase usually dominates the performance of whole inference procedure. Especially for Reddit dataset, aggregation phase occupies above 97% of the whole execution time. Even for the

This work is supported by National Natural Science Foundation of China (Grant No. 62072019, 61602022). Corresponding author is Jianlei Yang, Email: jianlei@buaa.edu.cn. The source code of this paper is publicly available on: https://github.com/BUAA-CI-Lab/GNN-Feature-Decomposition.

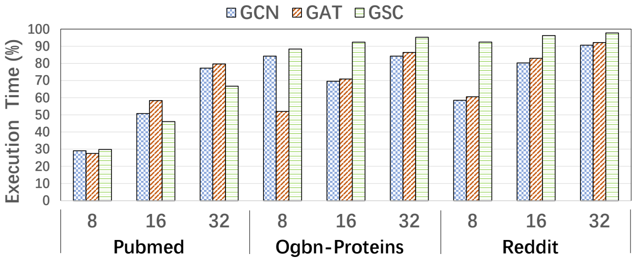

Fig. 1. Percentage of aggregation runtime comparison for GNNs inference (with hidden dimension of 8, 16 and 32).

TABLE I
GNNs Characterization on CPU.

	Aggregation	Combination
Exeucted IPC (IPC)	0.46	0.87
L1 Cache Miss Rate (L1 Miss)	41.86%	7.54%
LL Cache Miss Rate (LLC Miss)	45.59%	34.05%
TLB Cache Miss Rate (TLB Miss)	9.18%	0.15%
Data Reusability	Low	High
Execution Bound	Memory	Computation

small-scale Pubmed dataset, it still takes up more than half of the inference computation cost if the hidden dimension (output dimension of combination phase) exceeds 16.

Memory Access. Table I summarizes the execution patterns for GCN on Reddit, with the hidden dimension of 32. It is observed that **L1 Cache Miss Rate** and **Last Level Cache Miss Rate** in the aggregation phase are extremely high. Besides, the **Executed IPC** is only 0.46. The low efficiency of cache utilization in the aggregation phase is caused by the high randomness of memory accessing and poor data reuse between neighbor vertex. Only a few feature vectors can be loaded into the cache at the same time due to the high vertex feature dimension. Additionally, the peak memory usage during inference is approximately equal to 32 GB, resulting in a relatively high risk of OOM. Consequently, the aggregation phase of GNN is memory-bound, with irregular data access pattern and low data reusability.

B. Proposed Approach

To solve the above problems, we propose a new optimization method for GNN aggregation: **Feature Decomposition**, which can speed up GNN inference and avoid frequent OOM problem. Since the high dimension of feature data increases the reuse distance between vertices and causes OOM problems during aggregation phase, we decompose the dimension into smaller ones to alleviate these problems. In our method, the feature vector of each vertex is divided into $\mathcal{P}$ layers, and then the aggregation is performed layer by layer on all vertices. We take the computation of one vertex in a GNN layer as an example to describe our method.

As shown in Fig. 2, vertex Ⓐ (in red) in the graph has three neighbors: Ⓑ, Ⓒ, Ⓓ. General GNN computation on vertex Ⓐ can be summarized into two steps. Firstly, vertex Ⓐ aggregates its neighbors' feature vectors which are marked with blue, and then compute new activations of the vertex by MLP.

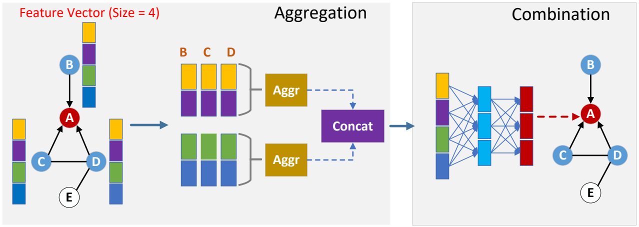

Fig. 2. Computation of one vertex (in red) in a GNN layer by feature decomposition method. The number of decomposition layers $\mathcal{P}$ is 2.

Algorithm 1: GNN Inference Procedure with Feature Decomposition

```
Input: Graph $G(V, E)$, Layers $\mathcal{K}$, vertex features $\boldsymbol{h}_v^{(0)}$ with
       length $L$ and the number of partitions $\mathcal{P}$.
Output: Updated vertex representations $\boldsymbol{z}_v$.
for $k = 1 \dots \mathcal{K}$ do
    /* Feature vector decomposition for all vertices */
    for $v \in V$ do
        $\boldsymbol{h}_v^{(k-1)}[\mathcal{P}] \leftarrow Chunk\left(\boldsymbol{h}_v^{(k-1)}, \mathcal{P}, \dim = -1\right)$
    end
    /* Aggregation on each decomposition layer */
    for $p \in \mathcal{P}$ do
        for $v \in V$ do
            $\boldsymbol{m}_{uv}^{(k)}[p] \leftarrow Message\left(\boldsymbol{h}_u^{(k-1)}[p], \boldsymbol{h}_v^{(k-1)}[p]\right)$
            $\boldsymbol{a}_v^{(k)}[p] \leftarrow Aggregate\left(\boldsymbol{m}_{uv}^{(k)}[p] \mid u \in N(v)\right)$
        end
    end
    $\boldsymbol{a}_v^{(k)} \leftarrow Concat\left(\boldsymbol{a}_v^{(k)}[0 : p-1]\right)$
    $\boldsymbol{h}_v^{(k)}[p] \leftarrow Update\left(\boldsymbol{a}_v^{(k)}, \boldsymbol{h}_v^{(k-1)}\right)$
end
$\boldsymbol{z}_v \leftarrow \boldsymbol{h}_v^{(\mathcal{K})}$
```

Unlike the general computation process, feature decomposition aims to decompose the feature vector before conducting the aggregation. In Fig. 2, we split the feature vector of all vertices into 2 layers, perform aggregation on each layer respectively, and finally concatenate the feature vectors together. After that, combination phase is performed to update the feature vector of vertex Ⓐ.

Algorithm 1 describes the complete computation procedure of GNN using our feature decomposition method. Same as the single node computation shown in Fig. 2, the feature vectors of all vertices are decomposed into $\mathcal{P}$ layers and aggregation is executed on these layers separately. In fact, even though feature decomposition brings efficiency improvements by executing aggregation separately, it usually still brings some additional overhead. Performing aggregation on each layer of feature vector requires repeated access to all edges. Therefore, the number of decomposition layers should be determined according to the memory capacity of the device and characteristics of datasets (edges, feature dimension, etc.).

C. Implementations with PyG Framework

Feature decomposition is implemented with PyG framework by slight modifications. PyG defines GNN execution paradigm

TABLE II
INFORMATION OF EVALUATED GRAPH DATASETS.

Name	# Vertices	# Edges	Avg. Degree
Pubmed	19,717	88,676	4.5
Ogbn-Proteins	132,534	39,561,252	597
Reddit	232,965	114,615,892	492

based on MessagePassing. The MessagePassing interface of PyG relies on a gather-scatter scheme to aggregate messages from neighboring vertices. Therefore, the input of the forward process is the edge set of graph in COO format and the feature vectors of all vertices. The aggregation process of GNN is performed by propagate. To provide users with a feature decomposition choice, we modified the forward function in the MessagePassing class. The specific modifications are mainly listed as follows:

- A new parameter is added to the forward function: Layers. It means how many layers the feature vector is divided into.
- In the forward function, the input feature vector is divided by torch.chunk referring to Layers, and the propagate is performed respectively on all layers.
- After the propagate is completed, the feature vectors of each layer is concatenated together by torch.concat.

III. EXPERIMENTAL RESULTS

The proposed feature decomposition approach is evaluated and compared with two aggregation schemes in PyG on Intel CPU and Raspberry Pi. The evaluated graphs information are listed in Table II, where Pubmed [9] has fewer vertices and low average degree of vertex, Ogbn-Proteins [10] and Reddit [7] have larger data scale and higher average degree. The granularity of feature decomposition embodies in the dimension of each decomposition layer, which is set as {GCN: 1, GAT: 8, GSC: 4} for Reddit and Ogbn-Proteins, and {GCN: 4, GAT: 8, GSC: 4} for Pubmed. For GNN models, we select GCN [1], GSC [7] and GAT [8] to cover GNNs with different aggregate operators. Among the two schemes in PyG, the Memory Efficient Aggregation scheme PyG_MEA does not support the implementation of GAT, therefore, the corresponding data is omitted.

Learn from previous experience [11], we perform the combination phase ahead of aggregation, which usually helps to improve aggregation efficiency. As a result, the feature dimension in aggregation phase is determined by hidden dimension of combination. When the hidden dimension is set above 32, the peak memory usage during GNN inference exceeds the limit of 32 GB, thus causing OOM problem for all baseline methods. Therefore, we set the hidden dimension as {8,16,32} for comparison between different methods, while also implementing our method with higher hidden dimension for further evaluation.

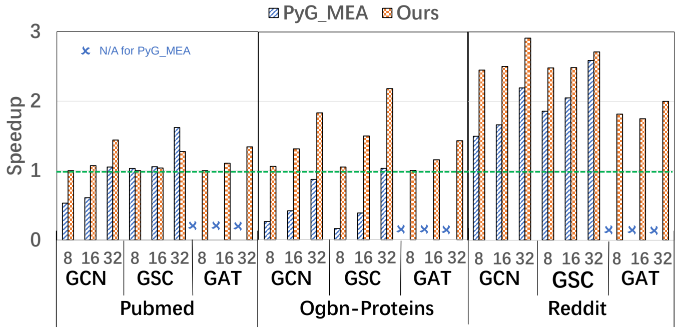

Fig. 3. Performance comparison between our proposed approach and PyG. The PyG with standard aggregation scheme is marked by the green dash line as a baseline. The PyG_MEA could obtain some speedups in Reddit dataset than baseline. Our approach could obtain significant speedups in many cases.

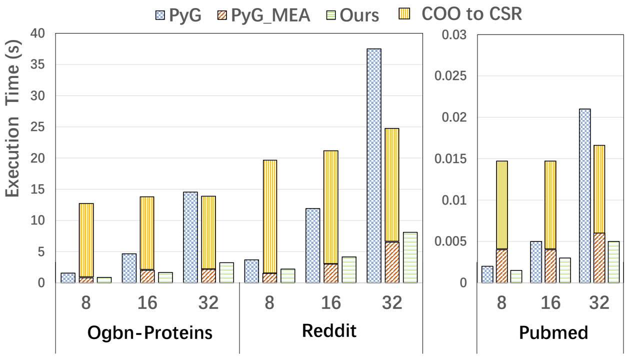

Fig. 4. Latency of aggregation and format conversion.

A. Inference Latency

As our first motivation is to reduce latency in GNN inference, here we first evaluate the latency of our feature decomposition method across several benchmarks. The relative end-to-end speedups over PyG under different configurations are summarized in Fig. 3. The results show that the speedup provided by our method is significant (up to 3×) and universal, with benefits on large graphs and high average degree.

PyG_MEA does not perform well on datasets with high degree and short lengh of feature vectors due to excessive format conversion overhead. As shown in Fig. 4, PyG_MEA takes nearly 13 seconds to perform format conversion on Ogbn-Proteins. The overhead cannot be offset by the improvement of its aggregation efficiency. In contrast, our method has no data conversion overhead and is more efficient.

B. Impact of Feature Decomposition Granularity

As we described earlier, feature decomposition has a tradeoff between feature data reusability and edge data utilization. Different decomposition layers are evaluated with various hidden dimensions as shown in Fig. 5. As the number of decomposition layers increases, the optimization brought by feature data reuse will be gradually neutralized with redundant memory access of edge data. Therefore, the number of decomposition layers should be selected reasonably to obtain the optimal efficiency in aggregation. Furthermore, with the expansion of the hidden dimension, our method can achieve

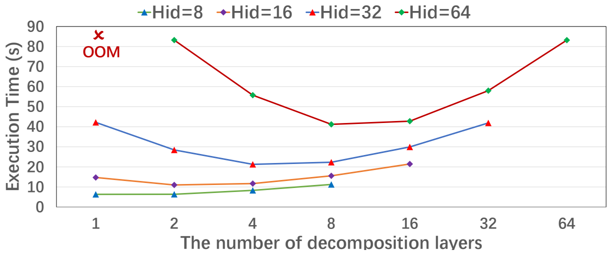

Fig. 5. Feature decomposition evaluation of GAT on Reddit with various decomposition layers and hidden dimensions. Hid means hidden dimension. × OOM means out-of-memory appeared when with Hid=64 and only 1 decomposition layer deployed.

TABLE III
PROFILING RESULTS OF GCN/GAT/GSC ON REDDIT WITH HID=32.

Model	Method	L1 Miss	LLC Miss	TLB Miss	IPC
GCN	PyG	23.15%	43.49%	4.43%	0.66
	Ours	**6.94%**	**5.09%**	**0.05%**	**1.82**
GAT	PyG	24.18%	43.01%	4.58%	0.63
	Ours	**10.39%**	**9.75%**	**0.44%**	**1.44**
GSC	PyG	40.47%	42.27%	8.69%	0.48
	Ours	**15.71%**	**3.48%**	**0.09%**	**1.25**

better speedup. In addition, as long as the aggregation phase can be executed when hidden dimension is 1, our method can avoid the OOM problem. As shown in Fig. 5, the baselines have OOM problem with hidden dimension 64. In contrast, GAT optimized by our method can still inference efficiently.

C. Memory Efficiency

Memory efficiency is evaluated by analyzing cache miss rate and peak memory usage. Comparison results between our approach and PyG are illustrated in Table III.

Cache Access Performance. Our method could significantly reduce the cache miss rate and improve executed IPC, because the reuse of neighbor feature data has been improved by reducing the feature vector length during aggregation phase.

Peak Memory Usage. Fig. 6 compares the peak memory usage of different aggregation schemes with hidden dimension Hid=32. It is obvious that our method can significantly reduce the peak memory usage, i.e., the memory efficiency is improved up to 5×. The GCN inference on Reddit with our method only requires 6 GB memory, which effectively alleviates the frequent OOM problem when performing GNN inference on the large-scale graphs.

D. Evaluations on Raspberry Pi Device

To further evaluate our method on resource-limited edge devices, we chose Raspberry Pi with only 1 GB memory as the experimental platform. Due to the extremely limited hardware resources, only Pubmed is evaluated and the speedups are shown in Fig. 7. As the hidden dimensions increasing, we could achieve better inference performance. For a very large hidden dimension, Hid=1024, PyG will has OOM while our approach could still works efficiently.

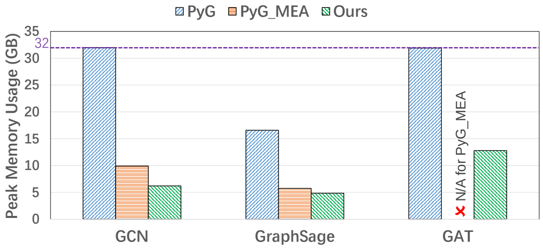

Fig. 6. Peak memory usage comparison for the first layer of GCN/GAT/GSC evaluated on Reddit, with hidden dimension Hid=32. PyG_MEA is not available for running GAT.

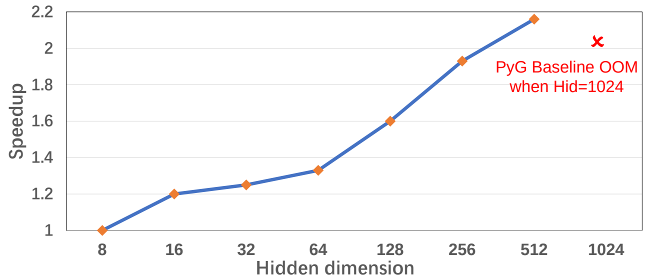

Fig. 7. Speedup of GCN with our method on Raspberry Pi compared with original PyG, where PyG has OOM when hidden dimension exceeds 1024.

IV. CONCLUSIONS

This work proposes feature decomposition for optimizing memory efficiency of GNN inference. Especially for the OOM problems in edge devices due to the very limited available hardware resources, it could significantly reduce the required peak memory. The results have shown that our method can provide a great boost for GNN edge computing applications.

REFERENCES

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[2] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of NIPS*, pages 5165–5175, 2018.

[3] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of CVPR*, pages 4938–4947, 2020.

[4] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation for graph neural networks. In *Proceedings of SIGKDD*, page 997–1005, 2020.

[5] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Machine Learning and Systems*, 2:187–198, 2020.

[6] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *Proceedings of ICLR*, 2019.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of NIPS*, page 1025–1035, 2017.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[9] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

[10] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[11] Mingyu Yan, Zhaodong Chen, Lei Deng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Characterizing and understanding GCNs on GPU. *IEEE Computer Architecture Letters*, 19(1):22–25, 2020.