# Single-GPU GNN Systems: Pitfalls and Future Directions

Yidong Gong and Pradeep Kumar, William & Mary

Numerous graph neural network (GNN) models, e.g., GCN [1], GAT [2], GIN [3], and others [4]–[6] have been proposed, leading to several GNN system-level optimizations [7]–[15], reportedly led up to 15× speedup in training runtime in single-GPU over a baseline DGL system. However, our analysis points out that many prior optimizations lack a comprehensive understanding of the unique requirement for GNN computations, which leads to pitfalls in system design and evaluation. In this work, we focus on *single-GPU GNN systems*, and analyze over 20 systems [7]–[22], [22]–[26] from the top systems and HPC conferences.

### A. Training Accuracy Related Pitfalls

**EVAL-P1: Absence of Training Accuracy Measurement** Several GNN training systems [7]–[11], [13]–[15], [18]–[20], [23], [27]–[29] have established a trend of not reporting training accuracy. Fig. 1 (a) shows that GNNAdvisor, Seastar, and TC-GNN demonstrate abnormal accuracy. FuseGNN [12] on GAT (not shown in Fig.) could achieve only 58% accuracy, way below DGL's 92.4% on the Reddit dataset. Further, a few other systems [8], [10] do not even have a backward computation, hence we cannot measure their training accuracy. Inspired by these results, further analysis shows that many GNN systems have the following system design pitfalls.

**SYS-P1: Omitted State Tensor** Many GNN systems [8], [10] fused their kernel into a giant kernel, utilizing their fused forward computation as a substitute for GNN training performance evaluation without implementing backward computation. However, these approaches often overlook the state tensors– intermediate activation results that require materialization (in GPU memory) for backward computation. Consequently, their emulated forward computation potentially fuse kernels by using shared memory or device registers for state tensor rather than global memory to gain a performance advantage, which is not practical in end-to-end training setup.

**SYS-P2: Missing/Inefficient Sparse Matrix Transpose** Forward SpMM ($Y \leftarrow AX$) is fundamentally different than SpMM$^T$ called in backward($\delta X \leftarrow A^T \delta Y$) because transpose need of A– the adjacency matrix (graph) that includes the static part(graph topology) and the dynamic part(the edge-level state tensor or weights). While, one can pre-process the graph to keep the transpose of the topology, the edge-level tensor requires transpose at run-time. Our measurement shows that CuSparse API performs transpose is even costlier than its SpMM runtime. DGL first transposes edge-level tensor using edge ID abstraction, which we call *eShuffle* in the poster, and then uses SpMM to implement SpMM$^T$. Hence, substituting forward SpMM in place of backward SpMM [14] is likely to show performance gain against DGL, which itself picked an inferior SpMM$^T$ design despite Cusparse offering it natively.

**SYS-P3: Incorrect Order of Backward Operations** A few GCN systems [7], [11] that attempt kernel fusion of SpMM with *divide by degree* kernel order their backward operation incorrectly, as the latter kernel should be called first in backward. However, they substitute forward fused version in backward leading to inaccurate accuracy, and some performance gain.

### B. Training Runtime Related Pitfalls

**EVAL-P2: Unawareness of Framework Overhead** Several single-GPU GNN systems [7], [9], [14], [25], [28] relied exclusively on smaller datasets, while others [11]–[13] relied additionally on one mid-size dataset to show performance speedup over DGL. However, that results were inconclusive due to DGL being shown as out-of-memory (OOM) or the host system being shown as slower. Our measurements for smaller datasets show that training time is dominated by framework overhead instead of kernel runtime. So, the training speedup stated in such cases is due to the lower framework overhead (unknowingly) instead of better kernel runtime (claimed). Fig. 1(b) and (c) confirm this empirically.

Our kernel runtime evaluation on mid-size datasets (see poster) confirms that pitfalls have impacted research: a) many current GNN systems' $\mathbf{SpMM}^\top$ (used in GAT) runtime is significantly slower than Cusparse; b) GNNAdvisor and TC-GNN were significantly slower than Cusparse for GCN SpMM.

**Conclusion and Future Work** In conclusion, new measurements presented in this paper question our understanding of single-GPU GNN system design and evaluation and motivate the need for more in-depth studies. Finally, we seek a benchmarking tool to truly compare the performance of prior works.

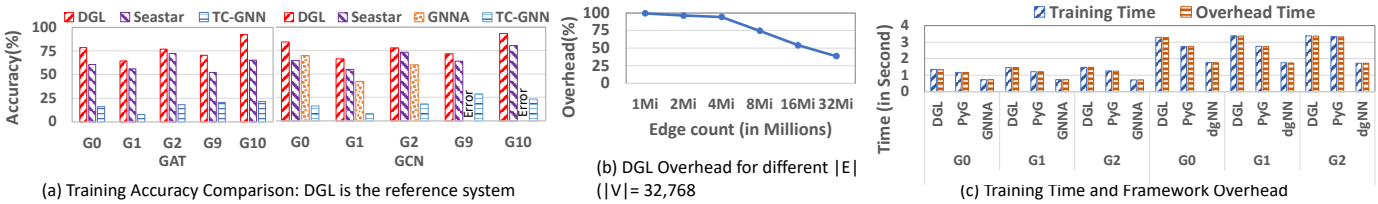


**Fig. 1:** Our evaluation results show pitfalls in GNN system. GNNA = GNNAdvisor. G0, G1, G2, G9, and G10 represent Cora, Citeseer, Pubmed, OGB product, and Reddit datasets respectively with vertex-count (edge-count) of 2,708(10,858), 3,327(9,104), 19,717(88,648), 2,449,029(123,718,280), 232,965(229,231,784) respectively

## REFERENCES


[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations (ICLR-17)*, 2017.

[2] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *7th International Conference on Learning Representations (ICLR-19)*, 2019.

[4] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[5] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2018, pp. 339–349.

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[7] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "Gnnadvisor: An adaptive and efficient runtime system for gnn acceleration on gpus," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 515–531.

[8] K. Huang, J. Zhai, Z. Zheng, Y. Yi, and X. Shen, "Understanding and bridging the gaps in current gnn performance optimizations," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 119–132.

[9] G. Huang, G. Dai, Y. Wang, and H. Yang, "Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–12.

[10] Q. Fu, Y. Ji, and H. H. Huang, "Tlpgnn: A lightweight two-level parallelism paradigm for graph neural network computation on gpu," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 122–134.

[11] Y. Wu, K. Ma, Z. Cai, T. Jin, B. Li, C. Zheng, J. Cheng, and F. Yu, "Seastar: Vertex-centric programming for graph neural networks," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 359–375.

[12] Z. Chen, M. Yan, M. Zhu, L. Deng, G. Li, S. Li, and Y. Xie, "fusegnn: Accelerating graph convolutional neural network training on gpgpu," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[13] H. Zhang, Z. Yu, G. Dai, G. Huang, Y. Ding, Y. Xie, and Y. Wang, "Understanding gnn computational graph: A coordinated computation, io, and memory perspective," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 467–484, 2022.

[14] Y. Wang, B. Feng, Z. Wang, G. Huang, and Y. Ding, "{TC-GNN}: Bridging sparse {GNN} computation and dense tensor cores on {GPUs}," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 149–164.

[15] Z. Ye, R. Lai, J. Shao, T. Chen, and L. Ceze, "Sparsetir: Composable abstractions for sparse compilation in deep learning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 660–678.

[16] Y. Hu, Z. Ye, M. Wang, J. Yu, D. Zheng, M. Li, Z. Zhang, Z. Zhang, and Y. Wang, "Featgraph: A flexible and efficient backend for graph neural network systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–13.

[17] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[18] A. Jangda, S. Polisetty, A. Guha, and M. Serafini, "Accelerating graph sampling for graph machine learning using gpus," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021.

[19] S. Liang, Y. Wang, C. Liu, L. He, L. Huawei, D. Xu, and X. Li, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," vol. 70, no. 9, pp. 1511–1525, 2020.

[20] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," in *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[22] C. Tian, L. Ma, Z. Yang, and Y. Dai, "Pcgcn: Partition-centric processing for accelerating graph convolutional network," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 936–945.

[23] I. Kim, J. Jeong, Y. Oh, M. K. Yoon, and G. Koo, "Analyzing gcn aggregation on gpu," *IEEE Access*, vol. 10, pp. 113 046–113 060, 2022.

[24] M. Yan, Z. Chen, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Characterizing and understanding gcns on gpu," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 22–25, 2020.

[25] T. Baruah, K. Shivdikar, S. Dong, Y. Sun, S. A. Mojumder, K. Jung, J. L. Abellán, Y. Ukidave, A. Joshi, J. Kim *et al.*, "Gnnmark: A benchmark suite to characterize graph neural network training on gpus," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 13–23.

[26] R. Waleffe, J. Mohoney, T. Rekatsinas, and S. Venkataraman, "Mariusgnn: Resource-efficient out-of-core training of graph neural networks," in *Eighteenth European Conference on Computer Systems (EuroSys' 23)*, 2023.

[27] L. Wang, Q. Yin, C. Tian, J. Yang, R. Chen, W. Yu, Z. Yao, and J. Zhou, "Flexgraph: A flexible and efficient distributed framework for gnn training," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 67–82.

[28] J. Hu, S. Qian, Q. Fang, Y. Wang, Q. Zhao, H. Zhang, and C. Xu, "Efficient graph deep learning in tensorflow with tf_geometric," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 3775–3778.

[29] B. Zhang, R. Kannan, and V. Prasanna, "Boostgcn: A framework for optimizing gcn inference on fpga," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 29–39.