Proceedings of the 4th International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/2026.38493

A Survey on Computational Efficiency Optimization for Graph Neural Networks on Large-Scale Graphs

Jiahao Wan

Faculty of Science and Engineering, University of Nottingham Ningbo China, Ningbo, China
3268446729@qq.com

Abstract. Graph Neural Networks (GNNs) have become a dominant paradigm for learning from graph-structured data, yet their application to large-scale graphs remains constrained by three coupled system bottlenecks: memory, computation, and communication. This paper organizes the literature around a bottleneck-centric taxonomy that maps representative techniques, including Cluster-GCN, GraphSAINT, GraphSAGE, SGC, DistDGL, and AliGraph, to the specific constraint each primarily relieves. Three representative systems, GraphSAGE, Cluster-GCN, and DistDGL, are further analyzed as case studies and consolidated into a practical decision matrix linking graph scale, hardware budget, and structural properties to an appropriate strategy. The paper also identifies a largely unaddressed challenge: the shift toward high-dimensional, LLM-encoded node features, which is beginning to move the dominant bottleneck from computation toward memory bandwidth and I/O. The review concludes that current sampling and partitioning techniques remain vulnerable to long-tail bias, that reported near-linear distributed speedups depend on network conditions rarely disclosed in published work, and that the most promising direction for future systems lies not in a new algorithm but in adaptive schedulers that reconfigure strategy at runtime.

Keywords: Graph Neural Networks, Computational Efficiency, Large-Scale Graphs, Distributed Training

1. Introduction

Graph Neural Networks (GNNs) have emerged as a powerful tool for learning representations from graph-structured data, with applications spanning social network analysis, recommendation systems, and knowledge graph reasoning. Since the introduction of the Graph Convolutional Network (GCN) [1], GNNs have achieved remarkable success on tasks such as node classification, link prediction, and graph classification. However, as real-world graphs continue to grow to billions of nodes and edges, the message-passing mechanism at the core of GNNs, which requires aggregating information from a node's neighbors, leads to exponential neighborhood expansion, high memory consumption, and substantial communication overhead when training is distributed across multiple devices. These challenges have made computational efficiency a central concern in the deployment of GNNs on industrial-scale graphs.

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.38493

Representative efforts include GraphSAGE [2], which samples fixed-size neighborhoods; Cluster-GCN [3], which partitions graphs for mini-batch training; and DistDGL [4], which distributes training across machines. These show that gains can come from algorithmic redesign, partitioning, and systems engineering, each with distinct accuracy–scalability–complexity trade-offs.

This paper adopts a literature review methodology. It follows a bottleneck-oriented structure—memory, compute, and communication—and analyzes representative methods as case studies to compare trade-offs and derive practical guidance. It also considers an emerging shift: as node features increasingly come from LLM encoders rather than low-dimensional attributes, the dominant bottleneck may itself be changing. By consolidating a fragmented literature into a coherent taxonomy, this paper clarifies the current landscape and highlights open challenges for scalable graph learning systems.

2. A bottleneck-centric taxonomy

Existing approaches to improving GNN efficiency on large-scale graphs are often presented as a flat list of algorithmic tricks. A more useful organizing principle is to ask which system bottleneck a technique relieves: a memory wall, where activations and sampled subgraphs exceed available memory; a compute wall, where the exponential growth of the receptive field makes per-iteration computation prohibitive; and a communication wall, where moving partitions, features, or gradients across devices dominates training time. Table 1 summarizes this taxonomy.

Table 1. A bottleneck-centric taxonomy of GNN efficiency optimization techniques

Bottleneck	Manifestation	Representative Solutions
Memory Wall	Mini-batch or full-graph activations exceed GPU/host memory, causing out-of-memory failures	Cluster-GCN [3], GraphSAINT [5], Roc [6]
Compute Wall	Neighborhood explosion across layers makes per-iteration computation and redundant re-aggregation costly	GraphSAGE [2], FastGCN [7], SGC [8]
Communication Wall	Cross-device or cross-machine transfer of partitions, features, or gradients dominates wall-clock training time	DistDGL [4], AliGraph [9], NeuGraph [10]

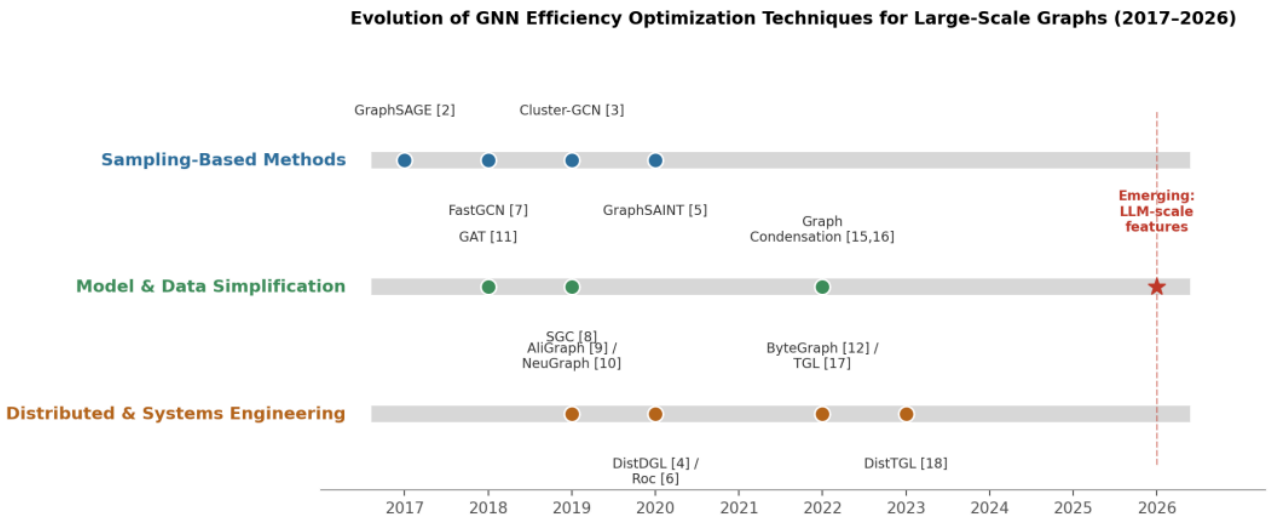


Figure 1. Evolution of GNN efficiency optimization techniques for large-scale graphs, 2017-2026, organized into three parallel research threads

DOI: 10.54254/2755-2721/2026.38493

2.1. The memory wall

Chiang et al.'s Cluster-GCN [3] relieves the memory wall by partitioning the graph into densely connected clusters and forming mini-batches from one or more clusters, bounding batch memory independently of graph size while preserving local connectivity. Zeng et al.'s GraphSAINT [5] instead samples subgraphs directly, using normalization to correct the resulting bias. Jia et al.'s Roc [6] adds a system-level memory manager that decides which activations to keep, recompute, or offload, jointly optimized with partitioning for accuracy. All three trade some statistical fidelity for a bounded, predictable memory footprint.

2.2. The compute wall

Hamilton et al.'s GraphSAGE [2] addresses the compute wall by sampling a fixed number of neighbors per layer, bounding per-node computation regardless of true degree and making the model inductive. Chen et al.'s FastGCN [7] instead applies layer-level importance sampling to reduce redundant computation shared across a mini-batch. Wu et al.'s SGC [8] takes the most aggressive route, removing nonlinearities between layers entirely and precomputing a single propagation matrix. By contrast, attention-heavy architectures such as GAT [11] sit at the opposite end of this trade-off and are poorly suited where the compute wall binds.

2.3. The communication wall

Zheng et al.'s DistDGL [4] relieves the communication wall by partitioning the graph across machines and co-locating sampling with the corresponding partition, together with workload balancing. Zhu et al.'s AliGraph [9] combines distributed storage, caching, and specialized sampling operators at industrial scale, while Ma et al.'s NeuGraph [10] targets multi-GPU communication within a single machine through chunk-based scheduling. More recent systems such as ByteGraph [12] and hybrid CPU-GPU training [13] extend this to industrial storage and billion-scale heterogeneous graphs, while PinSAGE [14] shows that production deployments often combine sampling with an efficient data pipeline to relieve multiple walls at once.

2.4. Emerging directions: graph condensation and dynamic graphs

Two directions depart from static per-batch training. Graph condensation [15, 16] optimizes a small synthetic graph to reproduce full-graph training, permanently reducing memory and compute, but is best validated on homogeneous graphs. Dynamic graphs compound all three bottlenecks; TGL [17] and DistTGL [18] show that static-graph techniques require redesign for temporal and distributed settings.

3. Case analysis

This section examines GraphSAGE [2], Cluster-GCN [3], and DistDGL [4] as exemplars of sampling-, partition-, and system-level distributed optimization. GraphSAGE samples a fixed number of neighbors per layer, decoupling per-batch cost from graph size and enabling inductive learning, but its sampling variance hurts high-degree nodes. Cluster-GCN builds mini-batches from densely connected clusters, preserving neighborhood structure and improving gradient quality, though its benefit depends on partition quality. DistDGL distributes graph, features, and computation across machines and reduces communication via workload balancing, at the cost of deployment

DOI: 10.54254/2755-2721/2026.38493

complexity. The three are largely composable: distributed systems can incorporate sampling. Table 2 summarizes practical selection guidance and common pitfalls.

Table 2. A practical decision matrix linking deployment scenarios to recommended strategies and pitfalls

Deployment Scenario	Recommended Approach	Pitfall to Avoid
Extremely large graph (>1 billion edges), limited to a single GPU	Subgraph sampling, e.g., GraphSAINT [5]	Avoid full distributed systems such as DistDGL [4]; overhead outweighs benefit at this scale
Graph with strong community structure	Partition-based batching, e.g., Cluster-GCN [3]	Avoid naive sampling such as GraphSAGE [2], which can disrupt community connectivity
Multiple machines available, relatively static graph	Distributed training with sampling, e.g., DistDGL [4]	Avoid full-batch training on the complete graph; infeasible at scale
Very high-dimensional node features (e.g., LLM-encoded embeddings)	Feature-simplification approaches, e.g., SGC [8]	Avoid attention-heavy architectures such as GAT [11]; memory-intensive at high dimensionality

4. Discussion

Across the literature, efficiency and accuracy remain in tension: aggressive simplification such as SGC [8] reduces cost but may lose capacity for higher-order structure. Algorithmic and system-level optimizations are largely complementary: sampling and simplification reduce per-step computation, while distributed systems such as DistDGL [4] and AliGraph [9] scale that computation across hardware; the most effective deployments, such as PinSAGE [14], typically combine both layers. Generalizability remains open, since partition-based methods such as Cluster-GCN [3] perform well on community-structured graphs but less so on sparse or heterogeneous ones, and most benchmarks cover a narrow set of standard datasets.

Practically, this matters for industry deployments at billion-edge scale: engineering choices—such as PinSAGE's data pipeline [14] and Roc's memory management [6]—can matter as much as algorithmic innovation, suggesting that GNN efficiency is as much a systems problem as a machine learning one.

A challenge absent from the works above is the shift toward LLM-encoded node features, which commonly reach thousands of dimensions and can make feature retrieval and storage costlier than aggregation itself, shifting the binding constraint toward memory bandwidth and I/O. None of GraphSAGE [2], Cluster-GCN [3], or DistDGL [4] were evaluated under such regimes. Dimensionality reduction before propagation is a straightforward mitigation, though it risks discarding semantic information; a more promising, largely unexplored direction is dimension-aware sampling, where sampling probability in methods such as GraphSAINT [5] is adjusted by feature dimensionality and retrieval cost, not just graph structure, to balance I/O against information gain.

5. Conclusion

This study set out to examine the strategies used to improve the computational efficiency of Graph Neural Networks on large-scale graphs. Organizing the literature around a bottleneck-centric taxonomy, rather than a flat list of algorithms, surfaces three observations that a more conventional survey structure tends to obscure.

First, sampling is not a free lunch: sampling-based techniques such as GraphSAGE and GraphSAINT are typically evaluated on aggregate accuracy metrics that can mask degradation for

DOI: 10.54254/2755-2721/2026.38493

long-tail, low-degree nodes, an unresolved weakness shared across nearly all sampling-based methods.

Second, distributed training does not guarantee linear speedup. The near-linear scalability reported for systems such as DistDGL typically holds only under favorable network and workload-balance conditions rarely disclosed in full, and in bandwidth-constrained environments communication overhead can offset much of the computational gain.

Third, the most promising direction for future work is unlikely to be another point-wise algorithm. Since the appropriate strategy depends on graph scale, structure, hardware budget, and, increasingly, feature dimensionality, as illustrated by the emerging LLM-embedding challenge, future systems would benefit more from an adaptive scheduler that reconfigures strategy at runtime than from a single new sampler or partitioning heuristic.

This research contributes a bottleneck-oriented taxonomy and a practical decision matrix that offer a more actionable reference than a purely method-by-method summary. The analysis is limited by its reliance on a literature review rather than original experiments and by the relatively small set of systems examined; a systematic empirical comparison under standardized memory, communication, and feature-dimensionality budgets would be a natural next step. Overall, this study suggests that future large-scale GNN systems will be defined less by any single optimization technique and more by the ability to adapt among them.

References

[1] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[2] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C.-J. (2019). Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining (KDD)*.

[4] Zheng, D., Ma, C., Wang, M., Zhou, J., Su, Q., Song, X., Gan, Q., Zhang, Z., & Karypis, G. (2020). DistDGL: Distributed graph neural network training for billion-scale graphs. In *Proc. IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*.

[5] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2020). GraphSAINT: Graph sampling based inductive learning method. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[6] Jia, Z., Lin, S., Gao, M., Zaharia, M., & Aiken, A. (2020). Improving the accuracy, scalability, and performance of graph neural networks with Roc. In *Proc. Machine Learning and Systems (MLSys)*.

[7] Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[8] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019). Simplifying graph convolutional networks. In *Proc. Int. Conf. on Machine Learning (ICML)*.

[9] Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y., & Zhou, J. (2019). AliGraph: A comprehensive graph neural network platform. *Proc. VLDB Endowment, 12*(12).

[10] Ma, L., Yang, Z., Miao, Y., Xue, J., Wu, M., Zhou, L., & Dai, Y. (2019). NeuGraph: Parallel deep neural network computation on large graphs. In *Proc. USENIX Annual Technical Conference (ATC)*.

[11] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[12] Li, C., Chen, H., Zhang, S., Hu, Y., Chen, C., Zhang, Z., Li, M., Li, X., Han, D., Chen, X., et al. (2022). ByteGraph: A high-performance distributed graph database in ByteDance. *Proc. VLDB Endowment, 15*(12), 3306-3318.

[13] Zheng, D., Song, X., Yang, C., LaSalle, D., & Karypis, G. (2022). Distributed hybrid CPU and GPU training for graph neural networks on billion-scale heterogeneous graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 4582–4591).

DOI: 10.54254/2755-2721/2026.38493

[14] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*.

[15] Jin, W., Zhao, L., Zhang, S., Liu, Y., Tang, J., & Shah, N. (2022). Graph condensation for graph neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[16] Jin, W., Tang, X., Jiang, H., Li, Z., Zhang, D., Tang, J., & Yin, B. (2022). Condensing graphs via one-step gradient matching. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)* (pp. 720–730).

[17] Zhou, H., Zheng, D., Nisa, I., Ioannidis, V., Song, X., & Karypis, G. (2022). TGL: A general framework for temporal GNN training on billion-scale graphs. *Proceedings of the VLDB Endowment, 15*(8).

[18] Zhou, H., Zheng, D., Song, X., Karypis, G., & Prasanna, V. (2023). DistTGL: Distributed memory-based temporal graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.