

Quantization-Aware Acceleration of Graph Neural Networks on GPUs

Jussi Hannula

Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA.
jussi.hannula578@buffalo.edu

Veit Hertrampf

Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV,
USA.

veithertrampf931@unr.edu

Abstract

Graph neural networks have become a central computational model for relational data, spanning recommendation, chemistry, transportation, social analysis, and scientific discovery. Their deployment on graphics processing units has enabled substantial progress, yet graph workloads remain difficult to accelerate because they combine irregular memory access, dynamic neighbor sampling, sparse matrix operations, and dense neural computation. Quantization-aware acceleration offers a powerful but structurally complicated response to these pressures. By reducing numerical precision in representations, activations, messages, and gradients, quantization can lower memory bandwidth demand, increase arithmetic throughput, and make larger graphs tractable on fixed hardware. However, the benefits are not automatic. Quantized graph neural networks introduce new trade-offs among accuracy, robustness, fairness, compiler support, runtime scheduling, multi-GPU communication, and long-term infrastructure sustainability. This paper examines quantization-aware acceleration as a system-level problem rather than a purely algorithmic compression technique. It analyzes how low-precision arithmetic interacts with GPU kernel design, memory hierarchies, graph sampling, distributed training, and deployment governance. It also considers how quantization decisions affect vulnerable populations when graph models are used in high-stakes domains, and how documentation, auditing, and policy frameworks must adapt. The discussion concludes that effective quantization for graph neural networks requires co-design across hardware, compilers, runtime systems, model architectures, and institutional governance. Without such integration, quantization may deliver local efficiency gains while producing hidden costs in accuracy, inequity, and operational fragility.

Keywords

graph neural networks; quantization; GPU acceleration; systems co-design; sustainable artificial intelligence; robustness; fairness; infrastructure governance.

1. Introduction

Graph neural networks have emerged as a dominant approach for learning from relational data. Early formulations of graph neural computation established the idea that node representations can be refined through repeated message passing over graph structure [4]. Subsequent architectures such as graph convolutional networks and graph attention networks demonstrated that graph learning could be integrated with modern deep learning pipelines [1], [2]. Inductive methods further expanded the scope of graph neural networks by allowing

models to generalize to unseen nodes and subgraphs [3]. Today, comprehensive surveys describe a broad ecosystem of architectures, training objectives, and application domains [5], [6]. This growth has made graph neural networks a critical workload for both cloud-scale systems and edge deployments.

The computational character of graph neural networks differs sharply from that of convolutional or transformer models. Graph data is irregular, often power-law distributed, and frequently too large to fit comfortably in device memory. Message passing requires repeated gathers and scatters across adjacency structures, producing memory access patterns that are difficult to coalesce on graphics processing units. Sampling-based training adds further irregularity because each minibatch may touch a different subset of vertices and edges. At the same time, the dense neural components within graph neural networks benefit from the same tensor cores, high-bandwidth memory, and parallel execution resources that have driven progress in mainstream deep learning. Acceleration therefore requires simultaneous attention to sparse graph operations and dense numerical kernels.

Quantization has become one of the most promising levers for improving the efficiency of deep learning on graphics processing units. Reducing numerical precision can shrink memory footprints, reduce bandwidth consumption, and allow hardware to process more operations per cycle [11], [12], [13]. These benefits are especially attractive for graph neural networks because memory movement, not pure arithmetic, often dominates runtime. Yet quantization in graph settings is not simply a matter of replacing floating-point values with lower-bit integers. Graph neural networks propagate information through neighborhoods, so quantization error can be amplified through multiple hops. Rare nodes, high-degree hubs, and minority groups may experience different error profiles. The result is a socio-technical problem that involves architecture, systems engineering, fairness, and governance.

This paper argues that quantization-aware acceleration of graph neural networks on graphics processing units should be understood as a system-level design problem. It requires coordination among kernel design, memory hierarchy management, graph partitioning, sampling strategies, distributed communication, compiler transformations, and deployment policy. The paper first reviews background and motivation, then examines quantization as a system lever. It subsequently analyzes GPU execution, memory and communication trade-offs, robustness and fairness, deployment and sustainability, evaluation methodologies, and open challenges. The final section concludes with a research and governance agenda for reliable quantized graph learning.

2. Background and Motivation

Graph neural networks generalize deep learning to domains where entities and relations are first-class data. In a typical message-passing formulation, each node maintains a hidden representation that is updated by aggregating information from its neighbors. Graph convolutional networks simplify this process through spectral-inspired neighborhood averaging [1]. Graph attention networks introduce learned attention weights that allow nodes to prioritize different neighbors [2]. Inductive graph models such as GraphSAGE sample and aggregate fixed-size neighborhoods, making training feasible on large graphs [3]. Earlier recurrent graph neural models established foundational ideas about state propagation and convergence [4]. Together, these approaches form the architectural basis for many modern graph learning systems.

The application landscape is correspondingly broad. Graph neural networks are used for recommendation, fraud detection, drug discovery, traffic forecasting, knowledge graph completion, and social network analysis. Surveys of the field emphasize that graph learning is not a single task but a family of tasks involving node classification, link prediction, graph classification, and generative modeling [5], [6]. This diversity matters for quantization because different tasks tolerate numerical error differently. A recommendation model may tolerate modest ranking degradation, while a molecular property predictor or medical graph model may require stringent numerical fidelity. The same quantization policy can therefore produce very different operational consequences across domains.

The systems motivation for quantization is straightforward. Graph neural network training and inference are often memory-bound. Embedding tables, adjacency structures, sampled subgraphs, and intermediate activations compete for limited high-bandwidth memory. Reducing precision can decrease the bytes moved per edge traversal and per feature aggregation. It can also increase effective arithmetic throughput when hardware supports low-precision matrix operations. However, graph neural networks are not merely dense matrix multiplication workloads. They include sparse operations, irregular indexing, dynamic control flow, and graph sampling. Quantization must therefore be integrated with the full execution pipeline rather than applied only to dense layers.

This integration challenge explains why graph neural network acceleration has attracted attention from both machine learning and computer architecture communities. Hardware acceleration studies have explored specialized datapaths, memory scheduling, and graph-aware execution [8]. Runtime systems such as GNNAdvisor adapt execution to the structure of graph workloads on graphics processing units [9]. Caching systems such as PaGraph address the problem of scaling training on large graphs through computation-aware caching [10]. These efforts show that graph learning performance depends on a delicate balance between locality, parallelism, and data movement. Quantization adds another dimension to that balance.

3. Quantization as a System Lever

Quantization reduces the numerical precision used to represent weights, activations, gradients, messages, and sometimes graph embeddings. In mainstream deep learning, reduced precision training and inference have been studied extensively. Early work on limited numerical precision showed that deep networks can remain accurate when arithmetic is carefully constrained [11]. Integer-arithmetic-only inference demonstrated that quantized networks could be deployed efficiently on integer hardware while preserving competitive accuracy [12]. Practical whitepapers described how convolutional networks can be quantized through calibration and rounding [13]. Hardware-aware automated quantization further proposed mixed precision policies tuned to device constraints [14]. More recent work on eight-bit floating-point formats has expanded the design space for accelerated training and inference [15].

For graph neural networks, quantization can be applied at multiple levels. Node features, edge features, adjacency weights, attention coefficients, and intermediate messages can each be quantized separately. Layers may use different precision levels depending on their sensitivity. Gradient communication in distributed training can be compressed through quantized all-reduce operations. Graph sampling may operate on quantized embeddings or produce quantized subgraphs. This flexibility is attractive because graph neural networks have heterogeneous computational phases. Neighborhood aggregation is often sparse and memory-

bound, while feature transformation is dense and compute-heavy. Mixed precision can allocate low precision to tolerant phases and higher precision to sensitive ones.

The benefits of quantization are primarily systemic rather than merely model-centric. Lower precision reduces memory capacity pressure, which is crucial when graphs and embeddings exceed device memory. It reduces bandwidth demand, which is often the limiting factor for message passing. It can improve cache utilization because more values fit into the same cache lines. It can increase tensor core utilization when matrix operations are expressed in supported low-precision formats. It can also reduce energy consumption per operation because lower-precision arithmetic and data movement require less switching energy. These gains are especially valuable in large-scale training and in edge deployment where thermal and power constraints are severe.

However, quantization introduces structural trade-offs. Post-training quantization is simple but may fail on out-of-distribution nodes or high-degree vertices. Quantization-aware training can recover accuracy but increases training cost and complexity. Mixed precision requires sensitivity analysis, calibration data, and runtime dispatch. Quantized kernels may require packing, alignment, and dequantization steps that offset some of the theoretical savings. In graph settings, irregular memory access can prevent the full bandwidth benefit that quantization provides in dense workloads. If a kernel is latency-bound rather than bandwidth-bound, reducing precision may not improve performance proportionally. The decision to quantize is therefore a system optimization problem involving accuracy targets, hardware capabilities, workload structure, and deployment constraints.

4. GPU Execution and Kernel Design for Quantized GNNs

Graphics processing units are well suited to the dense portions of graph neural networks but less naturally suited to irregular graph traversal. A modern GPU contains many streaming multiprocessors, a deep memory hierarchy, shared memory, register files, and specialized matrix units. Graph neural network kernels must map message passing, neighbor aggregation, and feature transformation onto these resources. Sparse matrix multiplication, gather-scatter operations, and sampling kernels often dominate runtime. Hardware acceleration research has explored specialized graph processing units and memory-aware scheduling to improve these operations [8]. Runtime systems have adapted kernel launch parameters, tiling, and partitioning to graph structure [9]. Caching systems have reduced redundant memory traffic for large graphs [10].

Quantized kernel design adds several constraints. Low-precision values must be loaded, unpacked, and converted into arithmetic formats supported by tensor cores or integer units. If the graph adjacency is quantized, indexing and masking must preserve structural correctness. If messages are quantized, aggregation must manage overflow and rounding. If gradients are quantized, optimizer updates may require scaling factors to avoid underflow. These operations create new instruction overhead that can be hidden only if the kernel has sufficient parallelism. For irregular graphs, divergence across warps may prevent efficient use of low-precision matrix units. Recent work on extremely fast GPU kernel design for graph neural network training illustrates how specialized scheduling and kernel organization can address some of these bottlenecks [7]. The broader lesson is that quantization cannot be treated as a standalone pass; it must shape the kernel interface itself.

Kernel fusion is one promising direction. Instead of writing quantized messages to global memory and then reading them back for aggregation, fused kernels can combine sampling,

message construction, quantization, and reduction. This reduces memory traffic and amortizes conversion overhead. However, fusion complicates resource management because registers and shared memory are limited. It may also reduce occupancy if fused kernels become too large. Another direction is precision-aware tiling. Dense feature blocks can use low precision and tensor cores, while sparse adjacency blocks use integer indexing and compressed representations. The system must decide which tiles use which format, and that decision may depend on runtime graph structure.

Compiler support is essential. Deep learning compilers such as TVM provide graph-level optimization, scheduling, and hardware-specific code generation [19]. Frameworks such as PyTorch provide flexible tensor operations and autograd support that can be extended to quantized graph layers [20]. Yet graph neural networks introduce dynamic shapes, irregular control flow, and sparse tensors that challenge existing compiler abstractions. A quantization-aware compiler for graph neural networks must reason about precision propagation, layout selection, kernel fusion, and distributed communication. Without such support, practitioners may rely on hand-tuned kernels that are difficult to port across GPU generations. This fragility is a major obstacle to sustainable deployment.

5. Memory Hierarchy, Graph Sampling, and Communication Trade-offs

Memory hierarchy management is central to quantized graph acceleration. Graph neural networks repeatedly access node features, edge lists, adjacency structures, and sampled subgraphs. Large graphs often exceed device memory, forcing out-of-core execution or distributed partitioning. Caching systems such as PaGraph show that computation-aware caching can reduce remote memory access and improve training throughput [10]. Quantization interacts with caching because smaller values increase effective cache capacity and reduce transfer volume. A cache that stores quantized embeddings can hold more nodes than a full-precision cache. However, quantized values may need scaling factors or zero-point metadata, which add overhead. If metadata is stored per block or per tensor, it may be manageable; if stored per element, it erodes the benefit.

Sampling is another critical interaction. Many graph neural network training pipelines sample neighbors to control minibatch size and memory consumption. Sampling changes the distribution of messages and can interact with quantization error. If rare neighbors are sampled infrequently, their quantized representations may not be calibrated well. If high-degree hubs are sampled frequently, their quantization error may dominate aggregation. Adaptive sampling may need to consider numerical sensitivity as well as computational cost. A node with unstable activations might require higher precision or more careful rounding. This suggests that sampling policies and quantization policies should be co-designed rather than optimized independently.

Distributed training introduces communication trade-offs. In multi-GPU systems, gradients, embeddings, and sampled features must be exchanged across devices. Quantization can reduce communication volume, which is often a bottleneck in large-scale training. However, compressed gradients may introduce bias or variance that slows convergence. Error feedback and stochastic rounding can mitigate these effects but add complexity. Communication-aware runtime systems such as GNNAdvisor demonstrate the importance of adapting execution to graph partitioning and network topology [9]. Quantized communication must be integrated with partitioning strategies so that low-precision exchanges do not create new load imbalances. If one partition contains many high-degree nodes, its quantized gradients may be noisier or larger after compression.

Robustness under skew is a persistent concern. Real-world graphs often have heavy-tailed degree distributions. A small number of vertices may connect to a large fraction of edges. These hubs are computationally expensive and numerically influential. Quantizing their messages can produce disproportionate error because their aggregated signals affect many neighbors. Conversely, low-degree nodes may be underrepresented in calibration data. A quantization policy that minimizes average error may therefore perform poorly for tails of the degree distribution. System designers must consider worst-case and subgroup-aware metrics, not only mean accuracy. This connects quantization to broader questions of robustness, fairness, and governance.

6. Robustness, Fairness, and Governance

Quantized graph neural networks are not socially neutral. Graph data often encodes human relationships, institutional histories, and structural inequalities. Fairness research in graph mining has shown that graph structure can amplify bias through homophily, community formation, and unequal visibility [18]. Quantization can interact with these dynamics in subtle ways. If minority subgroups occupy sparsely connected regions of the graph, their representations may receive less training signal and more quantization noise. If high-degree nodes dominate aggregation, quantization errors at hubs may propagate widely. If calibration data underrepresents certain communities, mixed precision policies may allocate lower precision to their features. The result can be a system that appears efficient in aggregate but degrades disproportionately for vulnerable groups.

Privacy considerations also arise. Differential privacy provides formal guarantees for releasing information about individuals in statistical datasets [16]. Quantization is sometimes mistakenly treated as a privacy mechanism because it reduces numerical detail. In practice, quantized embeddings and gradients can still leak information through membership inference, reconstruction, or linkage attacks. Quantization may change the attack surface, but it does not replace privacy engineering. Graph systems that combine quantization with federated learning or decentralized training require careful threat modeling. The interaction among low precision, secure aggregation, and graph structure is an open area for both systems research and policy analysis.

Governance frameworks must therefore extend beyond accuracy and latency. Datasheets for datasets encourage documentation of composition, collection, and intended use [17]. Similar documentation is needed for quantization policies: which precision levels were used, how calibration data was selected, what fairness metrics were monitored, and what failure modes were observed. Model cards and audit trails should record quantization decisions alongside architecture and training data. Procurement standards for public-sector graph systems should require evidence that quantized models do not systematically disadvantage protected groups. Without such governance, quantization can become an invisible source of inequity hidden behind impressive efficiency claims.

Policy implications extend to liability and transparency. If a quantized graph model is used in credit, employment, policing, or healthcare, affected individuals may have limited ability to contest decisions influenced by low-precision approximations. Regulators may need to ask whether quantization was part of a reasonable engineering process and whether its risks were assessed. Technical standards bodies could define reporting formats for precision, calibration, and subgroup performance. Professional societies could develop review guidelines for graph learning systems in high-stakes domains. These measures are not obstacles to innovation; they are conditions for trustworthy deployment.

7. Deployment, Sustainability, and Infrastructure Policy

Deployment contexts for quantized graph neural networks range from cloud data centers to edge devices and embedded systems. Cloud deployments benefit from reduced memory and bandwidth costs, enabling larger graphs or higher throughput per accelerator. Edge deployments benefit from lower power consumption and smaller memory footprints, which can allow graph learning on mobile devices, vehicles, or sensor networks. However, hardware fragmentation complicates deployment. Different GPUs support different low-precision formats, tensor core operations, and integer instruction sets. A model quantized for one architecture may not run efficiently on another without recompilation or retuning. Frameworks such as PyTorch and compilers such as TVM help manage portability, but graph-specific quantization remains underdeveloped [19], [20].

Sustainability is a first-order concern. Training large graph neural networks consumes substantial energy, and inference at scale can dominate operational carbon over time. Quantization can reduce energy per operation and per byte moved, which supports sustainable artificial intelligence goals. Yet the full lifecycle matters. Quantization-aware training may require additional retraining cycles, hyperparameter searches, and calibration runs. If these costs are ignored, the net energy benefit may be smaller than expected. Hardware refresh cycles also matter. If quantized models are tightly coupled to a specific GPU generation, they may become obsolete quickly, contributing to electronic waste. Sustainable design should favor portable quantization abstractions, reusable calibration pipelines, and hardware-aware but not hardware-locked kernels.

Infrastructure policy must address procurement, measurement, and accountability. Organizations that purchase GPU clusters for graph learning should require energy reporting and utilization metrics, not only peak throughput. Data centers should track the carbon intensity of workloads and support scheduling that shifts flexible graph training to low-carbon periods. Public research funding could encourage reproducibility by requiring release of quantization configurations, benchmark scripts, and hardware details. International coordination may be needed for standards around low-precision numerical formats, especially as vendors introduce competing eight-bit and four-bit representations [15]. Without coordination, fragmentation may slow adoption and increase costs.

Deployment also raises questions about centralized versus distributed governance. A cloud provider may control the hardware, runtime, and quantization policy, while an application developer controls the model and data. Responsibility for fairness or robustness failures may be ambiguous. Contracts and service-level agreements should specify precision guarantees, audit rights, and incident reporting. For public-sector deployments, transparency requirements may need to be stronger. The technical community can help by developing standard logs that record quantization parameters, calibration datasets, and subgroup evaluations. These logs would support both engineering debugging and external oversight.

8. Evaluation Methodologies and Cross-Domain Comparisons

Evaluating quantized graph neural networks requires more than top-line accuracy. System evaluations should report latency, throughput, memory footprint, energy consumption, and scalability across graph sizes. Model evaluations should report accuracy, calibration, robustness to adversarial or noisy edges, and subgroup fairness. These dimensions interact. A model may achieve high average accuracy with lower energy but perform poorly on low-degree nodes. A kernel may achieve high throughput on synthetic graphs but fail on real-

world power-law graphs. Benchmarks should therefore include diverse graph structures, dynamic sampling workloads, and realistic feature distributions. Surveys of graph neural networks provide a useful taxonomy of tasks and datasets [5]. Systems papers such as GNNAdvisor and PaGraph illustrate the value of end-to-end evaluation on large graphs [9], [10].

Cross-domain comparisons reveal different sensitivity profiles. Recommendation graphs are often bipartite and extremely large, with many low-degree users and a few popular items. Quantization may reduce embedding memory substantially, but popularity bias can be amplified if popular items receive lower precision. Molecular graphs are small but feature-rich; quantization may affect subtle chemical properties, making higher precision necessary for certain layers. Transportation graphs are spatial and temporal, with periodic patterns that may tolerate aggressive quantization in some regions but not others. Social graphs combine high-degree hubs, community structure, and sensitive attributes, making fairness evaluation essential. A single quantization recipe is unlikely to serve all domains well.

Comparisons with other accelerators also matter. Application-specific integrated circuits and field-programmable gate arrays can be highly efficient for fixed graph operations, and hardware acceleration research has explored such designs [8]. Graphics processing units remain attractive because they are programmable, widely available, and supported by mature software ecosystems. Quantization strengthens the GPU case by closing some of the efficiency gap with specialized hardware. However, if quantization requires extensive hand-tuning, the productivity advantage of GPUs may diminish. The optimal balance depends on deployment scale, model churn, and available engineering expertise.

Evaluation should also consider failure modes. A quantized graph system may behave well under average load but fail under adversarial edge insertion or degree skew. It may be robust to random noise but fragile to targeted perturbations. It may pass aggregate fairness tests but fail intersectional subgroup tests. Reporting only mean accuracy and peak throughput hides these risks. Researchers should adopt stress tests, ablation studies, and uncertainty quantification. Policy makers should demand independent audits for high-stakes deployments. These practices would improve both scientific understanding and public trust.

9. Open Challenges and Research Agenda

The first open challenge is unified quantization for irregular graphs. Existing quantization methods often assume dense tensors and regular computation. Graph neural networks require precision policies that adapt to sparse structure, dynamic sampling, and heterogeneous node degrees. Future work should develop sensitivity-aware quantizers that allocate precision per layer, per feature group, or even per vertex cluster. Such policies must be trainable, stable, and compatible with GPU kernels. They must also be explainable enough for auditors to understand why certain components use higher precision.

The second challenge is compiler and runtime co-design. Quantization decisions cannot be made independently of memory layout, kernel fusion, and distributed scheduling. Compilers should expose precision as a first-class optimization dimension, while runtimes should adapt quantization based on observed graph structure and hardware counters. This requires new intermediate representations for quantized graph tensors, new cost models for mixed precision, and new autotuning strategies. Frameworks and compilers provide a foundation, but graph-specific extensions are needed [19], [20]. The required reference at [7] demonstrates that

kernel-level innovation remains essential, but it must be connected to higher-level compilation and runtime control.

The third challenge is fairness-aware quantization. Fairness metrics must be integrated into quantization-aware training and calibration. Calibration datasets should be checked for representation bias. Mixed precision policies should be evaluated for subgroup error. If a particular community receives systematically lower precision, the system should flag or correct that outcome. This may require new optimization objectives that balance energy, latency, and equity. It also requires collaboration with domain experts who understand the social meaning of graph edges and attributes [18].

The fourth challenge is privacy and governance. Differential privacy offers rigorous foundations, but its integration with quantized graph training is not well understood [16]. Datasheets and documentation practices provide starting points for transparency [17]. Future work should develop standardized reports for quantization provenance, calibration data, and subgroup performance. Policy frameworks should clarify liability when quantization contributes to harm. International standards could reduce fragmentation and improve comparability across vendors and jurisdictions.

The fifth challenge is sustainability. Quantization can reduce operational energy, but lifecycle analysis must include training, calibration, hardware manufacturing, and disposal. Research should quantify these trade-offs across deployment scenarios. Infrastructure policy should reward measurable efficiency and penalize premature obsolescence. The goal is not merely faster graph neural networks but graph learning systems that are efficient, robust, fair, and accountable over their full service life.

10. Conclusion

Quantization-aware acceleration of graph neural networks on graphics processing units is a promising but deeply systemic endeavor. It can reduce memory pressure, bandwidth demand, and energy consumption, enabling larger graphs and broader deployment. Yet its benefits depend on careful coordination among model architecture, numerical precision, kernel design, memory hierarchy, sampling strategy, distributed communication, compiler support, and runtime adaptation. Quantization is not a simple compression pass; it changes the behavior of message passing and can amplify error in structurally uneven graphs. As graph neural networks are increasingly used in high-stakes settings, these technical choices become matters of robustness, fairness, privacy, and governance.

This paper has argued for a research agenda that treats quantization as a socio-technical system property. GPU kernel innovation remains essential, especially for irregular training workloads. Runtime systems, caching strategies, and distributed communication must be co-designed with precision policies. Evaluation must move beyond average accuracy to include subgroup performance, stress tests, energy accounting, and lifecycle sustainability. Documentation and policy frameworks must make quantization decisions visible to auditors and affected communities. Without these measures, quantization may deliver narrow efficiency gains while hiding significant costs. With them, it can contribute to graph learning systems that are faster, more sustainable, and more trustworthy.

References

1. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations.

2. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.
3. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* (pp. 1024–1034).
4. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61–80.
5. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1), 4–24.
6. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1, 57–81.
7. Peng, H., Xie, X., Shivdikar, K., Hasan, M. A., Zhao, J., Huang, S., ... & Ding, C. (2024, April). Maxk-gnn: Extremely fast gpu kernel design for accelerating graph neural networks training. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (pp. 683-698).
8. Dave, S., Baghdadi, R., Nowatzki, T., Avancha, S., Shrivastava, A., & Li, B. (2020). Hardware acceleration of graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)* (pp. 1–6). IEEE.
9. Wang, Y., Feng, B., Li, G., Xie, S., Li, Y., Ding, Y., & Xie, Y. (2021). GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (pp. 515–531).
10. Lin, Z., Li, C., Miao, Y., Liu, Y., & Xu, Y. (2020). PaGraph: Scaling GNN training on large graphs via computation-aware caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (pp. 401–415).
11. Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In *International Conference on Machine Learning* (pp. 1737–1746). PMLR.
12. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704–2713).
13. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*.
14. Wang, K., Liu, Z., Lin, Y., Lin, J., & Han, S. (2019). HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8612–8620).
15. Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellempudi, N., Oberman, S., Park, S., Root, B., & Siu, M. (2022). FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.
16. Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4), 211–407.

17. Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, 64(12), 86–92.
18. Kang, J., & Tong, H. (2023). Fairness in graph mining: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(10), 10588–10609.
19. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., & Krishnamurthy, A. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (pp. 578–594).
20. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (pp. 8024–8035).