

Dynamic Graph Neural Network Acceleration with Runtime-Aware GPU Kernels

Sai Mukherjee

Department of Computer Science, University of Alabama at Birmingham, Birmingham, AL, USA.

smukherjee@uab.edu

Lara McElgunn

Department of Computer Science, Binghamton University, Binghamton, NY, USA.

lmcelgunn@binghamton.edu

Opung Damanik

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

damanikopung@unr.edu

Abstract

Dynamic graph neural networks have become a central computational abstraction for modeling relational systems whose entities, edges, and features change over time. Although early graph neural network research established strong algorithmic foundations for representation learning on static graphs, the acceleration of dynamic graph workloads on graphics processing units remains constrained by irregular memory access, rapidly shifting sparsity patterns, temporal locality, and workload imbalance. This paper examines runtime-aware GPU kernel acceleration as a system-level response to those constraints. Rather than treating kernel optimization as a purely local performance problem, the discussion frames runtime-aware acceleration as an architectural and infrastructural practice that couples graph partitioning, memory hierarchy management, scheduling, precision selection, and temporal windowing. The paper analyzes structural trade-offs among throughput, latency, energy use, portability, and robustness, and it considers how deployment decisions influence fairness, accountability, and policy outcomes in socio-technical infrastructures. Through cross-domain comparisons involving transportation, finance, public health, and communication networks, the paper argues that dynamic graph neural network acceleration must be understood as a governance-sensitive infrastructure problem. The conclusion emphasizes that sustainable acceleration requires co-design across algorithms, runtimes, hardware, and institutional oversight, and that performance claims should be evaluated alongside reproducibility, energy intensity, and equitable access to computational capacity.

Keywords

dynamic graph neural networks, GPU acceleration, runtime systems, kernel specialization, graph systems, sustainable computing, fairness, governance.

1. Introduction

Graph neural networks have emerged as a powerful family of learning architectures for relational data because they combine local message passing with learned representations that respect graph structure. Foundational work established that neural message passing could

approximate graph functions and support classification, regression, and representation learning on vertices and edges [1]. Subsequent convolutional and inductive formulations extended these ideas to semi-supervised learning and large-scale graph sampling, making graph neural networks practical for citation networks, social platforms, recommendation systems, and molecular graphs [2], [3]. Comprehensive surveys have since documented the rapid expansion of graph neural network variants, training objectives, and application domains [4]. Yet the dominant research narrative has often treated graph learning as an algorithmic problem, leaving the computational infrastructure that makes large-scale deployment possible under-examined.

Dynamic graph neural networks intensify this infrastructure challenge because graph structure is no longer a fixed input. Edges appear and disappear, vertex features evolve, neighborhoods shift, and temporal ordering becomes semantically important. A model that performs well on a static benchmark may fail when deployed on a streaming financial transaction graph, a transportation network with time-varying congestion, or a public health contact graph that changes by the hour. Runtime-aware GPU kernels are therefore necessary not merely to speed up matrix operations but to adapt computation to changing graph topology and changing memory behavior. Recent kernel-level work has shown that specialized GPU designs can substantially accelerate graph neural network training by reorganizing work around graph sparsity and hardware execution constraints [5]. That line of inquiry raises broader questions about how runtime intelligence should be integrated into graph systems without sacrificing correctness, fairness, or maintainability.

This paper argues that dynamic graph neural network acceleration should be approached as a socio-technical systems problem. Runtime-aware kernels are not isolated artifacts; they are embedded within libraries, schedulers, deployment pipelines, monitoring systems, and institutional practices. Their design affects who can train models, how quickly decisions can be made, how much energy is consumed, and which populations are represented in the underlying graph. The paper develops this argument through several connected themes. It first examines the nature of dynamic graph workloads and the sources of runtime variability. It then analyzes the architecture of runtime-aware GPU kernels and the structural trade-offs they introduce. It subsequently considers infrastructure, governance, robustness, fairness, and policy implications. Finally, it offers cross-domain illustrations and forward-looking perspectives.

2. Dynamic Graph Workloads and Runtime Variability

Dynamic graph workloads differ from static graph workloads in ways that directly affect GPU kernel design. In static graphs, preprocessing can often produce stable partitions, fixed adjacency structures, and predictable sparsity patterns. In dynamic graphs, however, the active subgraph may change across iterations, batches, or time windows. Edges may be added or removed, vertex degrees may shift dramatically, and neighborhoods may exhibit bursty temporal behavior. These properties create runtime variability that undermines static kernel configurations. A kernel optimized for one sparsity regime may become inefficient when the graph becomes denser, when hot vertices accumulate most edges, or when temporal locality causes irregular memory access. Libraries such as PyTorch Geometric and Deep Graph Library provide abstractions that simplify graph learning, but they also reveal the tension between flexible graph representation and low-level kernel specialization [6], [7].

Temporal graph learning frameworks further illustrate this tension. PyTorch Geometric Temporal offers spatiotemporal signal processing primitives that support recurrent, attention-

based, and convolutional models over dynamic graphs [8]. EvolveGCN treats graph convolutional parameters as evolving over time, allowing the model to adapt to changing graph structure rather than assuming a fixed adjacency matrix [9]. Inductive representation learning on temporal graphs emphasizes that embeddings must generalize to unseen nodes and edges while respecting temporal order [10]. Temporal graph networks provide a general framework for continuous-time dynamic graphs by combining memory modules with message passing over temporal neighborhoods [11]. These approaches demonstrate that temporal semantics are not an optional layer added to a static pipeline; they change the data layout, the batching strategy, the sampling procedure, and the kernel launch configuration.

Runtime variability also arises from the interaction between graph sampling and GPU execution. Neighborhood sampling is often used to control fan-out and memory consumption, but sampled subgraphs can vary widely in size and shape. If sampling is performed on the central processing unit and then transferred to the graphics processing unit, transfer overhead and synchronization costs may dominate. If sampling is performed on the graphics processing unit, the sampler itself becomes a kernel that competes for memory bandwidth and compute resources. The runtime must therefore decide when to sample, where to sample, how many neighbors to retain, and whether to cache historical neighborhoods. These decisions are not purely mechanical. They determine the statistical properties of the training signal and can affect model quality, especially for rare or minority subgroups whose connectivity patterns differ from the majority.

The temporal dimension adds further complexity because graph updates may arrive as continuous streams rather than discrete batches. A deployment system may need to process event streams with low latency while also maintaining historical context for model training. This dual requirement creates a tension between online inference and offline learning. Runtime-aware kernels must support fast updates to vertex representations, efficient temporal neighborhood retrieval, and consistent handling of out-of-order events. When these capabilities are absent, practitioners often respond by simplifying the graph, reducing temporal resolution, or discarding long-range dependencies. Such simplifications may improve throughput but can also obscure structural inequities, rare events, and causal relationships that matter in high-stakes domains.

3. Architecture of Runtime-Aware GPU Kernels

Runtime-aware GPU kernel acceleration for dynamic graph neural networks requires a layered architecture that connects graph semantics to hardware execution. At the lowest level, kernels must manage memory access patterns that are irregular and data-dependent. At the middle level, a runtime scheduler must select among alternative kernel implementations, partition strategies, and precision modes. At the upper level, the graph learning framework must expose enough information about topology and temporal structure for the runtime to make informed decisions. This layered view distinguishes runtime-aware acceleration from static kernel optimization. Static optimization assumes a known workload shape, whereas runtime-aware acceleration assumes that the workload will change and that the system must adapt without violating correctness or reproducibility.

One important architectural direction is adaptive work assignment. Dynamic graphs often exhibit power-law degree distributions and temporal hotspots, meaning that a small number of vertices or time intervals account for a large share of computation. A runtime-aware system can rebalance work across streaming multiprocessors by splitting high-degree vertices, aggregating low-degree vertices, or switching between vertex-centric and edge-centric

execution. Research on graph convolutional network accelerators has explored runtime workload rebalancing to mitigate imbalance in sparse matrix operations [14]. Hybrid accelerator designs combine different execution engines for dense and sparse phases, reflecting the observation that graph neural network layers can alternate between compute-intensive and memory-intensive behavior [15]. These hardware-oriented insights are relevant to GPU runtimes because they highlight the need for dynamic phase detection and resource allocation.

Another architectural concern is memory hierarchy management. Dynamic graph neural networks frequently access vertex features, edge features, adjacency lists, temporal encodings, and intermediate embeddings. These structures have different reuse patterns. Some vertex features are accessed repeatedly within a time window, while others are accessed once and discarded. A runtime-aware kernel can promote frequently accessed features into shared memory or cache, while streaming less reusable data through global memory. It can also compress sparse structures, reorder vertices by locality, or fuse message passing with aggregation to reduce intermediate writes. Work on redundancy-free computation for graph neural networks shows that eliminating redundant neighborhood aggregation can substantially reduce memory traffic and computation [13]. Such techniques are especially valuable for dynamic graphs because they reduce the cost of repeated processing across time steps.

Runtime-aware GPU kernels also depend on scheduling and synchronization policies. Dynamic graph workloads often involve multiple kernels for sampling, message passing, aggregation, and parameter updates. If these kernels are launched sequentially with global synchronization between them, latency can be high and resource utilization can be low. If they are overlapped aggressively, race conditions and nondeterminism may arise. A runtime system must therefore choose between conservative and aggressive schedules based on graph size, temporal window, and hardware capacity. Adaptive runtime systems for graph neural network acceleration on GPUs have demonstrated that memory access patterns, workload balance, and kernel configuration can be tuned at runtime to improve performance across different graphs [12]. Surveys of graph neural network accelerators further show that the design space spans algorithmic transformations, compiler techniques, and hardware specialization [16]. The required kernel-level study cited here reinforces the importance of co-designing graph sparsity, GPU thread mapping, and training loop organization [5].

4. Structural Trade-offs and System-Level Constraints

The design of runtime-aware GPU kernels involves unavoidable trade-offs. The first is between specialization and portability. A kernel highly specialized for one graph distribution, hardware generation, or temporal pattern may achieve excellent performance on that target but fail to transfer to other settings. This is a significant problem for research reproducibility and industrial deployment. Organizations often operate heterogeneous GPU clusters, and models may move from development to edge inference. A runtime-aware system that relies on many hardware-specific assumptions may become brittle. By contrast, a more portable design may sacrifice peak performance but gain maintainability. The optimal balance depends on the deployment context and the cost of retuning.

The second trade-off is between latency and throughput. Dynamic graph applications such as fraud detection, traffic routing, and content moderation may require low-latency responses. Batch-oriented training, however, favors high throughput. Runtime-aware kernels can switch between modes, but doing so introduces complexity. A low-latency kernel may process a small temporal window with limited neighborhood expansion, while a high-throughput kernel

may use larger batches and deeper sampling. The choice affects model accuracy because temporal context and neighborhood size influence predictive quality. A system that dynamically chooses between these modes must therefore account for statistical as well as computational consequences. It cannot treat latency as a purely operational metric.

The third trade-off is between energy efficiency and computational intensity. Larger batches, wider neighborhoods, and higher precision generally improve training stability and model quality, but they consume more energy and memory bandwidth. Runtime-aware kernels can reduce precision, prune edges, or skip redundant computation, but these choices may introduce bias or instability. Research on energy and policy considerations in deep learning has shown that computational efficiency is not only a technical matter but also an environmental and institutional one [18]. Dynamic graph neural networks compound this issue because they may retrain or update continuously, turning a one-time training cost into an ongoing operational expense. Sustainable acceleration therefore requires runtime policies that consider energy budgets, carbon intensity, and hardware lifespan.

The fourth trade-off is between centralized and distributed execution. A single GPU offers simpler synchronization and lower communication overhead, but it may not accommodate very large dynamic graphs or high event rates. Distributed execution enables scale but introduces network communication, partitioning complexity, and consistency challenges. Dynamic graphs are especially difficult to partition because topology changes over time. A partition that is balanced at one moment may become skewed at the next. Runtime-aware systems must therefore consider repartitioning, replication, and caching strategies. These strategies affect fault tolerance, recovery time, and cost. They also raise governance questions about data locality, cross-border transfer, and access control.

5. Infrastructure, Deployment, and Governance

Runtime-aware GPU acceleration is embedded in an infrastructure stack that includes hardware provisioning, cluster scheduling, model registries, data pipelines, monitoring, and incident response. In enterprise and public-sector settings, these components are often governed by policies that determine who can access data, how long logs are retained, and what level of explainability is required. A graph neural network trained on a dynamic contact network for public health may need strong privacy protections, while a graph neural network used for financial fraud detection may be subject to audit and reporting requirements. The runtime layer is not exempt from these obligations. If a kernel skips certain edges, caches stale features, or approximates temporal neighborhoods, those decisions can affect the model outputs that are later audited.

Deployment also shapes the reproducibility of dynamic graph research. Static graph benchmarks are difficult enough to reproduce when hardware and software versions vary. Dynamic graphs add temporal ordering, event stream timing, and sampling randomness. A runtime-aware system that adapts to hardware conditions may produce different execution traces on different machines, even if the logical model is the same. This does not necessarily invalidate the approach, but it requires careful documentation of runtime policies, random seeds, graph snapshots, and hardware configurations. Governance frameworks for algorithmic auditing emphasize the importance of end-to-end documentation and accountability [20]. In a dynamic graph setting, such frameworks must extend to runtime decisions that are often hidden beneath library abstractions.

Infrastructure governance must also address access equity. High-performance GPU resources are concentrated in well-funded institutions and large technology firms. If dynamic graph neural network acceleration depends on specialized kernels and premium hardware, then the benefits of improved performance may accrue unevenly. Smaller research groups, public agencies, and community organizations may lack the capacity to train, deploy, and maintain such systems. Open-source libraries and portable runtime abstractions can reduce this gap, but they do not eliminate it. Sustainable infrastructure policy should therefore consider shared computing facilities, transparency in benchmark reporting, and support for reproducible research on heterogeneous hardware. The goal is not merely faster kernels but a computational ecosystem in which dynamic graph learning can be scrutinized, reused, and governed.

6. Robustness, Fairness, and Policy Implications

Robustness in dynamic graph neural networks is closely related to runtime behavior. A model may be robust to adversarial perturbations in a static graph but fragile when the graph changes over time. Runtime-aware kernels that cache historical embeddings, truncate temporal windows, or approximate neighborhood sampling can introduce failure modes that are not visible in standard accuracy metrics. For example, a fraud detection system may miss a newly emerging pattern because the runtime prioritizes recent high-degree vertices. A public health contact tracing system may underrepresent isolated communities because their temporal neighborhoods are sparse and therefore cheaper to skip. These are not only algorithmic problems; they are system-level problems that arise from the interaction between data distribution, runtime policy, and deployment constraints.

Fairness considerations are equally important. Graph neural networks propagate information through connections, so disparities in graph structure can amplify disparities in outcomes. If minority subgroups have fewer connections, noisier features, or different temporal activity patterns, then runtime decisions that reduce computation for sparse regions may worsen representational gaps. Research on bias and fairness in machine learning has shown that technical interventions alone are insufficient when structural inequalities shape the data [17]. In dynamic graphs, fairness must be evaluated over time. A model may appear fair at one snapshot but become unfair as the graph evolves. Runtime-aware acceleration should therefore include monitoring for subgroup performance, temporal drift, and representation quality. It should also support the ability to reweight, resample, or adjust neighborhood expansion when disparities are detected.

Policy implications follow from these concerns. Governments and standards bodies increasingly expect automated systems to be explainable, contestable, and accountable. Dynamic graph neural networks complicate these expectations because their decisions depend on evolving relationships and runtime approximations. A policy framework that only examines model architecture or training data may miss the influence of kernel scheduling and memory management. Conversely, a framework that demands full transparency for every runtime decision may be impractical for low-latency systems. The challenge is to design governance mechanisms that are proportionate to risk. High-stakes domains may require auditable runtime logs, human review, and strict limits on approximation. Lower-risk domains may allow more aggressive optimization with periodic evaluation.

The principle that ethical principles alone cannot guarantee ethical artificial intelligence is relevant here [19]. Runtime-aware acceleration needs institutional mechanisms, not only good intentions. These mechanisms include independent evaluation, red-team testing, incident

reporting, and continuous monitoring. They also include procurement policies that reward energy efficiency, reproducibility, and equity rather than raw speed alone. If performance benchmarks become the dominant criterion, then dynamic graph systems may optimize for narrow technical metrics while neglecting broader social consequences. A more balanced evaluation framework would consider accuracy, latency, energy, fairness, robustness, and maintainability together.

7. Case Illustrations and Cross-Domain Comparisons

Transportation networks illustrate the value and difficulty of runtime-aware acceleration. Traffic graphs are highly dynamic, with congestion propagating across road segments and transit schedules shifting by time of day. A graph neural network used for travel-time prediction must process a streaming graph with strong temporal dependencies. Runtime-aware kernels can prioritize active road segments, compress inactive regions, and adjust neighborhood sampling during peak periods. However, the system must also remain robust to sensor failures and unexpected events. If runtime optimization discards rarely used roads, it may fail during emergencies when those roads become critical. Cross-domain comparison with communication networks shows a similar pattern. Network traffic graphs exhibit bursts and hot spots, but they also require strict latency guarantees. Runtime-aware kernels must therefore balance adaptability with predictability.

Financial transaction graphs provide another compelling case. Fraud detection and anti-money-laundering systems operate on dynamic graphs where new accounts, transactions, and relationships appear continuously. The graph is adversarial because bad actors deliberately change behavior to evade detection. Runtime-aware kernels that favor recent high-degree vertices may improve throughput but also create blind spots for low-volume, high-risk patterns. Temporal graph networks and inductive temporal representation learning are relevant because they can incorporate time-aware memory and generalize to unseen nodes [10], [11]. EvolveGCN similarly offers a way to update model parameters as the graph changes [9]. Yet the deployment of such models raises serious governance questions about false positives, financial exclusion, and due process.

Public health and social service networks introduce additional ethical complexity. Contact graphs and service referral networks are dynamic and sensitive. Runtime-aware acceleration can help public agencies respond quickly during outbreaks or crises, but it can also enable surveillance if not carefully governed. The same kernel that accelerates message passing over a contact graph could be repurposed for monitoring. Policy safeguards must therefore address data minimization, purpose limitation, and access control. PyTorch Geometric Temporal and related frameworks make it easier to build spatiotemporal models [8]. That ease of use is valuable for rapid response, but it also lowers the barrier to irresponsible deployment. Governance must evolve alongside tooling.

Hardware case studies reinforce the importance of adaptability. Accelerators that rebalance workload at runtime for graph convolutional networks show that imbalance is a first-order concern [14]. Hybrid architectures that separate dense and sparse phases demonstrate that no single execution strategy dominates across all graph layers [15]. These findings suggest that GPU runtimes for dynamic graphs should not commit to one kernel shape or scheduling policy. Instead, they should support a portfolio of strategies and select among them based on observable graph properties. The challenge is to make this selection process transparent, reproducible, and aligned with domain-specific constraints.

8. Forward-Looking Perspectives

The future of dynamic graph neural network acceleration will depend on co-design across algorithms, runtimes, hardware, and institutions. Algorithmic research can develop models that expose temporal structure in ways that are amenable to efficient execution. Runtime research can build adaptive schedulers that respond to graph changes without sacrificing determinism or debuggability. Hardware research can provide memory hierarchies and interconnects that better match irregular graph workloads. Institutional research can define evaluation standards that reward energy efficiency, fairness, and reproducibility. None of these efforts is sufficient alone. The required kernel-level work on fast GPU design for graph neural network training shows the potential of deep specialization [5], while adaptive runtime systems show the value of flexibility [12]. The next step is to integrate these approaches into a principled architecture that treats runtime policy as part of the model lifecycle.

One promising direction is the development of runtime contracts. A runtime contract would specify the approximations a system is allowed to make, the conditions under which it must fall back to exact computation, and the metrics it must report. For example, a contract might require that temporal windows not be truncated below a certain duration in high-stakes decisions, or that subgroup performance be monitored whenever sampling density changes. Such contracts would make runtime behavior auditable without requiring full transparency into every kernel launch. They would also provide a common language for engineers, domain experts, and regulators.

Another direction is sustainable benchmarking. Current graph neural network benchmarks often emphasize accuracy and speed while neglecting energy, memory, and hardware utilization. Dynamic graph benchmarks should include temporal split protocols, streaming workloads, and hardware-aware metrics. They should also report variance across runs because runtime adaptation can introduce nondeterminism. Surveys of graph neural network accelerators provide a foundation for organizing this evaluation space [16]. Energy and policy research offers a framework for connecting computational efficiency to environmental impact [18]. Fairness and auditing research provides methods for assessing whether performance gains are distributed equitably [17], [20]. Together, these perspectives can guide the development of acceleration techniques that are not only fast but also responsible.

Finally, the field should attend to the political economy of acceleration. Faster dynamic graph neural networks can enable beneficial applications in medicine, transportation, and public safety. They can also enable more intensive surveillance, financial extraction, and automated decision-making. The direction of technological development is not determined by hardware alone. It is shaped by funding priorities, procurement rules, open-source governance, and professional norms. Researchers who build runtime-aware kernels should therefore consider how their design choices affect the distribution of power and risk. A kernel that reduces energy consumption may also reduce access barriers. A kernel that prioritizes low latency may increase opacity. These are not secondary concerns; they are constitutive of the systems being built.

9. Conclusion

Dynamic graph neural network acceleration with runtime-aware GPU kernels addresses a genuine and growing computational need. Dynamic graphs are irregular, temporal, and continually changing, and static kernel optimization cannot fully accommodate their variability. Runtime-aware designs offer a path toward better utilization of GPU resources by

adapting memory management, work assignment, scheduling, and precision to observed graph conditions. Yet these benefits come with structural trade-offs. Specialization can reduce portability, low latency can conflict with throughput, energy efficiency can challenge model quality, and distributed execution can complicate consistency and governance. The paper has argued that these trade-offs should be evaluated as part of a socio-technical infrastructure rather than as isolated performance questions.

The implications extend beyond engineering. Dynamic graph neural networks are increasingly used in domains that affect liberty, health, finance, and public services. Runtime decisions about sampling, caching, and approximation can influence fairness, robustness, and accountability. Governance frameworks must therefore reach into the runtime layer, not only the model layer. They must support auditing, reproducibility, energy reporting, and equitable access to computational resources. The most promising path forward is co-design: algorithms that expose useful temporal structure, runtimes that adapt transparently, hardware that matches irregular workloads, and institutions that reward responsible performance. If pursued with care, runtime-aware GPU acceleration can make dynamic graph learning faster, more sustainable, and more trustworthy.

References

1. 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. <https://doi.org/10.1109/TNN.2008.2005605>
2. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th 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 30* (pp. 1024–1034). Curran Associates.
4. 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. <https://doi.org/10.1109/TNNLS.2020.2978386>
5. 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).
6. Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.
7. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., ... & Zhou, J. (2019). Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv*. <https://arxiv.org/abs/1909.01315>
8. Rozemberczki, B., Scherer, P., He, Y., Panagopoulos, G., Riedel, A., Astefanoaei, M., ... & Sarkar, R. (2021). PyTorch Geometric Temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM International*

Conference on Information and Knowledge Management (pp. 4564–4573).
<https://doi.org/10.1145/3459637.3482014>

9. Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., ... & Leiserson, C. (2020). EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04), 5363–5370.
<https://doi.org/10.1609/aaai.v34i04.5984>
10. Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). Inductive representation learning on temporal graphs. In *Proceedings of the 8th International Conference on Learning Representations*.
11. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. *arXiv*.
<https://arxiv.org/abs/2006.10637>
12. 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 *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation* (pp. 515–531).
13. Jia, Z., Lin, S., Ying, R., You, J., Leskovec, J., & Aiken, A. (2020). Redundancy-free computation for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 997–1005).
<https://doi.org/10.1145/3394486.3403142>
14. Geng, T., Li, A., Shi, R., Wu, C., Wang, T., Li, Y., ... & Herboldt, M. C. (2020). AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *Proceedings of the 53rd IEEE/ACM International Symposium on Microarchitecture* (pp. 922–936). <https://doi.org/10.1109/MICRO50266.2020.00079>
15. Yan, M., Deng, L., Hu, X., Liang, L., Feng, Y., Ma, X., ... & Xie, Y. (2020). HyGCN: A GCN accelerator with hybrid architecture. In *Proceedings of the 26th IEEE International Symposium on High Performance Computer Architecture* (pp. 15–29).
<https://doi.org/10.1109/HPCA47549.2020.00012>
16. Abadal, S., Jain, A., Guirado, R., López-Alonso, J., & Alarcón, E. (2021). Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys*, 54(9), 1–38. <https://doi.org/10.1145/3477141>
17. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys*, 54(6), 1–35.
<https://doi.org/10.1145/3457607>
18. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 3645–3650). <https://doi.org/10.18653/v1/P19-1355>
19. Mittelstadt, B. (2019). Principles alone cannot guarantee ethical AI. *Nature Machine Intelligence*, 1(11), 501–507. <https://doi.org/10.1038/s42256-019-0114-4>
20. Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., ... & Barnes, P. (2020). Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (pp. 33–44). <https://doi.org/10.1145/3351095.3372873>