

Hardware-Aware Graph Neural Network Acceleration on Modern GPUs

Noor Lambregt

Department of Computer Science, Binghamton University, Binghamton, NY, USA.
noormail@binghamton.edu

Jacopo Serao

School of Computing, Clemson University, Clemson, SC, USA.
serao1970@clemson.edu

Abstract

Graph neural networks have become a central computational method for learning over relational data, with applications spanning recommendation, molecular modeling, fraud detection, transportation, and scientific simulation. Although modern graphics processing units offer substantial arithmetic throughput and memory bandwidth, graph neural network workloads remain difficult to accelerate because their execution is dominated by irregular memory access, sparse communication, dynamic topology, and variable-degree aggregation. This paper examines hardware-aware graph neural network acceleration as a socio-technical systems problem rather than a narrow kernel optimization problem. It analyzes the structural trade-offs among graph representation, memory hierarchy use, kernel specialization, multi-GPU distribution, compiler scheduling, and deployment governance. The discussion emphasizes that acceleration choices influence not only latency and throughput but also reproducibility, robustness, fairness, energy sustainability, and policy accountability. Through cross-domain comparisons and infrastructure-level analysis, the paper argues that effective acceleration requires co-design across algorithms, hardware, runtime systems, and institutional practices. It concludes that the next generation of graph learning systems must treat efficiency, equity, and environmental responsibility as inseparable design constraints, especially as graph models move from research prototypes to critical infrastructure.

Keywords

graph neural networks; GPU acceleration; hardware-aware computing; graph systems; distributed training; memory hierarchy; sustainability; fairness; policy.

1. Introduction

Graph neural networks have emerged as a powerful family of machine learning models for structured data, where entities are represented as nodes and their relationships as edges. Foundational formulations of graph neural computation established the idea that node representations can be learned through iterative message passing over graph neighborhoods [1]. Subsequent architectures such as graph convolutional networks and graph attention networks demonstrated that relational inductive biases can support strong performance in semi-supervised classification, link prediction, and representation learning [2], [3]. These methods have since been extended into a broad ecosystem of models for heterogeneous graphs, temporal graphs, molecular graphs, and knowledge graphs [5], [6]. The common computational pattern is neighborhood aggregation followed by feature transformation, often repeated across multiple layers.

Modern graphics processing units are natural candidates for accelerating such computations because they provide massive parallelism, high memory bandwidth, and specialized tensor operations. Yet graph neural networks do not map cleanly onto the dense, regular execution patterns that GPUs were originally designed to optimize. Graph neighborhoods are irregular, node degrees vary widely, and memory accesses are often scattered across large adjacency structures. As a result, arithmetic units may remain underutilized while memory latency and synchronization overhead dominate execution time [7]. This mismatch between graph structure and hardware regularity is the central systems challenge in graph neural network acceleration.

This paper argues that hardware-aware acceleration must be understood as a layered problem. At the lowest level, it involves kernel design, memory layout, and instruction scheduling. At the middle level, it involves graph partitioning, batching, communication, and runtime orchestration. At the highest level, it involves infrastructure governance, energy accounting, fairness auditing, and policy constraints. A system that optimizes only the kernel may achieve impressive benchmark results while remaining fragile in production, inequitable in its effects, or unsustainable in its resource consumption. Conversely, a system that attends only to governance without hardware-aware engineering may fail to meet practical latency and cost requirements.

The analysis therefore proceeds from foundations to architecture, then to distributed deployment, compiler infrastructure, and socio-technical implications. The goal is not to propose a single accelerator design but to map the structural trade-offs that researchers and practitioners must confront when deploying graph neural networks on modern GPUs. In doing so, the paper treats acceleration as an infrastructural practice that shapes who can train graph models, where they can be deployed, and what environmental and social costs accompany their use.

2. Foundations of Graph Neural Networks and GPU Execution

Graph neural networks operate by propagating information along edges. In a typical layer, each node gathers feature vectors from its neighbors, combines them through a permutation-invariant operation such as summation or averaging, and then transforms the aggregated representation with learned parameters. Early graph neural network formulations showed that such recursive neighborhood aggregation can approximate graph-structured functions under suitable conditions [4]. Later work simplified and scaled this idea through localized convolutional filters [2] and attention-based weighting [3]. Inductive methods further enabled generalization to unseen nodes by learning aggregation functions rather than fixed node embeddings [3].

From a hardware perspective, these operations contain both dense and sparse components. Feature transformation is often a dense matrix multiplication that can exploit tensor cores and high-throughput floating-point pipelines. Neighborhood aggregation, by contrast, is sparse and irregular. It requires reading neighbor indices, gathering corresponding feature rows, and accumulating results. When node degrees follow a long-tailed distribution, a small number of high-degree nodes can generate disproportionate memory traffic, while many low-degree nodes create underutilized warps. This imbalance is not merely an implementation detail; it is a structural property of graph data that shapes the entire acceleration strategy [5], [6].

The roofline model provides a useful conceptual frame for understanding these constraints, even without resorting to formal equations. It distinguishes between workloads limited by

arithmetic throughput and those limited by memory bandwidth or latency [16]. Graph neural network layers frequently fall into the latter category because each aggregated feature element is used in relatively few arithmetic operations relative to the cost of fetching it. The effective performance therefore depends on cache behavior, memory coalescing, and the ability to reuse data across neighboring nodes. When adjacency lists are stored in compressed sparse formats, random access patterns can defeat cache locality and reduce effective bandwidth.

GPU execution adds another layer of complexity. Graphics processing units execute threads in warps or wavefronts, and divergence within a warp can serialize execution. In graph aggregation, neighboring nodes may reside in distant memory regions, and different nodes may have different numbers of neighbors. This causes divergent control flow and irregular memory transactions. Techniques such as node reordering, degree-aware batching, and adjacency sorting can improve locality, but they also introduce preprocessing costs and may alter numerical results if not handled carefully. Hardware-aware acceleration thus becomes a balancing act between graph preprocessing, kernel specialization, and runtime flexibility.

3. Hardware-Aware Design Space for GNN Acceleration

The design space for graph neural network acceleration can be organized around several structural trade-offs. One trade-off concerns graph representation. Dense adjacency matrices enable regular computation and efficient tensor-core use, but they waste memory and compute on absent edges. Sparse representations conserve memory but produce irregular access patterns. Hybrid approaches attempt to separate dense subgraphs from sparse regions, applying different kernels to each. The choice depends on graph density, feature dimensionality, batch size, and the relative cost of memory versus computation on the target GPU [7].

A second trade-off concerns sampling and batching. Full-neighborhood aggregation can be accurate but expensive for high-degree nodes. Neighbor sampling reduces computation and memory traffic, but it introduces variance and may weaken the model’s ability to capture long-range dependencies [3]. Layer-wise sampling, subgraph sampling, and historical embedding methods each have different implications for GPU utilization. Sampling can make execution more regular by capping neighborhood size, but it also changes the learning problem and may require careful tuning to preserve accuracy. In production systems, the choice of sampling strategy is therefore both a statistical and a hardware decision.

A third trade-off concerns kernel fusion and operator scheduling. Graph neural network layers often consist of aggregation, normalization, linear transformation, activation, and dropout. If each operation is launched as a separate kernel, intermediate tensors must be written to and read from global memory, increasing bandwidth pressure. Fusing operations can reduce memory traffic and kernel launch overhead, but fusion may reduce parallelism or complicate memory layout. Compiler frameworks such as TVM demonstrate that automated scheduling and operator fusion can substantially improve deep learning performance across diverse hardware targets [13]. For graph workloads, however, fusion must account for sparse gathers and irregular reductions that are less amenable to static optimization.

A fourth trade-off concerns precision and numerical robustness. Lower-precision arithmetic can increase throughput and reduce memory footprint, which is attractive for large graphs. Yet aggregation over many neighbors can amplify rounding errors, especially when graph structure is skewed or features have wide dynamic range. Mixed-precision strategies may preserve accuracy while still exploiting tensor cores, but they require careful scaling and

validation. Hardware-aware design must therefore consider not only peak throughput but also the stability of the learning process under reduced precision.

These trade-offs interact. A dense representation may pair well with tensor cores but poorly with sparse sampling. A fused kernel may reduce bandwidth but limit the ability to dynamically reorder nodes. A distributed partition may balance computation but increase communication. Hardware-aware acceleration is thus a multidimensional optimization problem in which improvements in one dimension can degrade another. The most robust systems are those that expose these trade-offs to the runtime and allow policies to adapt to graph properties and deployment constraints.

4. Memory Hierarchy, Irregularity, and Kernel Specialization

The memory hierarchy is the decisive battleground for graph neural network acceleration. High-bandwidth memory provides large capacity but high latency. L2 cache offers lower latency but limited capacity. Shared memory and registers provide the fastest access but require explicit management. Graph aggregation often reads neighbor features from high-bandwidth memory, accumulates partial results in registers or shared memory, and writes updated node features back to global memory. If neighbor indices are not sorted or partitioned effectively, each warp may issue many uncoalesced transactions, wasting bandwidth and stalling execution.

Kernel specialization is a common response to this irregularity. Instead of using a single aggregation kernel for all nodes, systems may classify nodes by degree and launch different kernels for low-degree, medium-degree, and high-degree groups. Low-degree nodes can be processed with simple thread-level parallelism, while high-degree nodes may require block-level reductions or warp-level coordination. Such specialization can improve load balance and memory coalescing, but it increases code complexity and may require preprocessing to assign nodes to bins. The benefit depends on the distribution of degrees and the cost of reordering.

A recent line of work on extremely fast GPU kernel design for graph neural network training illustrates the potential of aggressive specialization and memory-aware scheduling [15]. The broader lesson is that kernel design must be co-designed with graph preprocessing, because the layout of adjacency structures determines what a kernel can efficiently access. IO-aware attention mechanisms in transformer models provide a useful analogy: by restructuring computation around memory traffic rather than arithmetic alone, they achieve substantial gains on modern GPUs [14]. Graph acceleration follows a similar logic, except that the irregularity is not only in sequence length but in neighborhood structure.

Kernel specialization also affects robustness and portability. A kernel tuned for one GPU generation may perform poorly on another with different cache sizes, memory bandwidth, or warp scheduling. It may also behave differently when graph degree distributions shift, as they often do in dynamic graphs. Robust deployment therefore requires autotuning, fallback kernels, and performance monitoring. The system must detect when a specialized path is no longer beneficial and switch to a more general implementation. This adaptive behavior is essential for long-lived infrastructure, where hardware and data distributions evolve over time.

5. Distributed and Multi-GPU Deployment

As graph neural networks scale to billions of edges and hundreds of millions of nodes, single-GPU acceleration becomes insufficient. Distributed training introduces new dimensions of hardware-aware design, including graph partitioning, communication scheduling, and

memory balancing. DistDGL, for example, addresses distributed graph neural network training by partitioning graphs and coordinating sampling across workers [10]. The central challenge is that graph edges cross partition boundaries, creating remote accesses and communication overhead. A partition that balances node counts may not balance edge cuts, and a partition that minimizes communication may create computational hotspots.

Pipeline and data parallelism offer complementary strategies. Pipeline parallelism splits model layers across devices and streams micro-batches to overlap computation and communication [11]. Memory optimization techniques such as ZeRO reduce redundant parameter and optimizer state replication, enabling larger models on limited hardware [12]. These methods were developed primarily for dense neural networks, but they can be adapted to graph workloads with careful handling of sparse dependencies. For example, pipeline stages that include aggregation may become imbalanced because some graph partitions have more edges than others. Dynamic scheduling and work stealing can help, but they add runtime complexity.

Multi-GPU graph training also raises questions of consistency and reproducibility. If neighbor sampling is distributed across workers, different random seeds or communication orders can produce different mini-batches. This affects model convergence and makes it difficult to reproduce results across hardware configurations. Hardware-aware systems must therefore log sampling policies, partition metadata, and synchronization behavior. In regulated domains, such as finance or healthcare, reproducibility is not only a scientific norm but also a compliance requirement.

Infrastructure governance becomes relevant here. Distributed training consumes significant energy and requires data center capacity, cooling, and network bandwidth [17]. The choice of parallelism strategy therefore has environmental implications. A system that uses more GPUs to achieve lower latency may consume more total energy than a system that uses fewer GPUs with longer runtime. Conversely, delayed training may incur opportunity costs or reduce model freshness. These trade-offs cannot be resolved by hardware alone; they require institutional policies that weigh performance, cost, and sustainability.

6. Compiler, Runtime, and Infrastructure Governance

Compiler and runtime systems mediate between graph algorithms and GPU hardware. Frameworks such as PyTorch Geometric and Deep Graph Library provide abstractions for message passing, batching, and sampling [8], [9]. These frameworks lower the barrier to entry, but they also shape what optimizations are possible. If the abstraction hides memory layout or kernel selection, users may struggle to apply hardware-aware optimizations. If the abstraction exposes too many details, it becomes difficult to maintain portability across GPU generations.

Domain-specific compilers can help by separating algorithm specification from scheduling. TVM demonstrates that tensor programs can be optimized automatically for different hardware backends through scheduling primitives [13]. For graph neural networks, a compiler must reason about sparse tensors, irregular loops, and data-dependent control flow. It may need to generate specialized kernels for different degree classes or partition strategies. It must also manage memory allocation for intermediate neighbor features, which can be large and variable. The compiler therefore becomes a site of governance, because its default choices determine which optimizations are available to ordinary users.

Runtime governance concerns monitoring, versioning, and accountability. Machine learning systems accumulate hidden technical debt through entangled dependencies, configuration drift,

and undeclared consumers [21]. Graph systems add further complexity because graph structure may change over time, and models may be retrained on different snapshots. Model cards and documentation practices can improve transparency by recording intended use, performance characteristics, and limitations [20]. Fairness considerations are also relevant, because graph-based decisions can propagate structural bias through network effects [19]. A recommendation model trained on a biased social graph may amplify existing inequalities, even if the underlying GPU kernels are highly optimized.

Infrastructure governance must therefore connect hardware metrics to social outcomes. Performance counters, energy measurements, and fairness audits should be treated as first-class telemetry. If a graph neural network is deployed in a public service, such as benefits allocation or credit scoring, the system should document not only accuracy but also resource consumption and subgroup performance. This is a departure from traditional acceleration research, which often focuses on benchmark speedups. A socio-technical approach recognizes that acceleration is not neutral; it changes who can deploy models, how often they are updated, and what environmental burden they impose.

7. Robustness, Fairness, and Sustainability Implications

Robustness in graph neural networks has both algorithmic and hardware dimensions. Algorithmically, graph models are vulnerable to adversarial perturbations, noisy edges, and distribution shift. A small change in graph structure can alter neighborhood aggregation and change predictions. Hardware-aware acceleration can exacerbate this fragility if it relies on sampling or approximation that varies across runs. Conversely, robust design can be supported by hardware that enables ensemble methods, uncertainty estimation, or repeated sampling with controlled randomness.

Hardware faults and thermal effects also matter. Large-scale GPU training can experience memory errors, communication failures, or performance throttling. Checkpointing and fault-tolerant scheduling are common in distributed training, but graph-specific state includes not only model parameters but also graph partitions and sampler state. Recovery must restore these components consistently. If a partition is lost and recomputed differently, the resulting model may differ from the original run. Robustness therefore depends on reproducibility guarantees that span hardware, software, and data management.

Fairness in graph learning is deeply structural. Nodes are connected, so decisions about one node can affect others through message passing and downstream predictions. If a graph encodes historical discrimination, a graph neural network may learn and reinforce that pattern [19]. Hardware acceleration can indirectly affect fairness by enabling larger models, more frequent updates, or broader deployment. For example, a highly optimized GPU pipeline may make it feasible to score millions of users daily, increasing the scale of potential harm if the model is biased. Fairness audits must therefore consider the entire inference pipeline, not only the model architecture.

Sustainability is another cross-cutting concern. Training and deploying large neural networks consume substantial energy, and graph models can be especially demanding because of irregular memory access and distributed communication [17], [18]. Hardware-aware acceleration can reduce energy per inference or per training step, but it can also create rebound effects. If graph models become cheaper, organizations may train more of them or use larger graphs, increasing total consumption. Policy instruments such as energy reporting, carbon-aware scheduling, and procurement standards can help align acceleration research

with environmental goals. The challenge is to design incentives that reward both efficiency and sufficiency.

8. Case Illustrations and Cross-Domain Comparisons

Different domains illustrate different acceleration priorities. In recommendation systems, graphs are massive, dynamic, and skewed. The goal is often low-latency inference over billions of user-item interactions. Hardware-aware design may prioritize embedding table access, negative sampling, and cache-friendly partitioning. In molecular property prediction, graphs are smaller but feature-rich, and the goal is often high-throughput training over many molecules. Here, batching and tensor-core utilization may matter more than distributed partitioning. In transportation and power grids, graphs are spatial and temporal, and robustness to missing data or sensor failure is critical.

Scientific computing offers a useful comparison. Sparse linear algebra and particle simulations also face irregular memory access, and they have developed sophisticated techniques for graph partitioning, communication avoidance, and cache blocking. Graph neural network acceleration can learn from these communities, especially in the design of distributed sparse kernels and load balancing. At the same time, graph neural networks introduce learned parameters and stochastic sampling, which are less common in traditional simulation. The result is a hybrid workload that requires both numerical and machine learning expertise.

Cross-domain comparison also reveals governance differences. A social network recommender may be optimized for engagement, while a clinical decision support system must prioritize interpretability, privacy, and regulatory compliance. The same GPU kernel may be used in both, but the surrounding infrastructure and accountability mechanisms differ. Hardware-aware acceleration cannot be evaluated in isolation from these contexts. A speedup that is acceptable for advertising may be unacceptable for healthcare if it reduces auditability or increases inequity.

The scale of deployment matters as well. A single-GPU training system for academic research has different constraints from a multi-tenant cloud service serving millions of users. The former may prioritize flexibility and reproducibility, while the latter prioritizes throughput, isolation, and cost. Hardware-aware design must therefore be parameterized by deployment context. There is no universal optimal kernel or partitioning strategy. Instead, there are families of strategies whose suitability depends on graph size, degree distribution, latency requirements, privacy constraints, and energy budgets.

9. Open Challenges and Policy Implications

Several open challenges remain. Dynamic graphs require acceleration systems that can handle continual updates without full recomputation. Heterogeneous hardware requires portable abstractions that can target different GPU architectures, accelerators, and edge devices. Benchmarking remains fragmented, making it difficult to compare systems fairly. Many papers report speedups on specific datasets without disclosing preprocessing costs, energy consumption, or numerical precision. A more rigorous benchmarking culture would include end-to-end metrics, reproducibility artifacts, and sensitivity analysis.

Policy implications follow from these technical gaps. Public funding agencies may require energy and fairness reporting for graph learning projects that use substantial computational resources. Procurement standards can favor systems that expose telemetry and support

independent auditing. Data governance rules can constrain how graph data is partitioned and shared across jurisdictions, affecting distributed training architectures. Privacy regulations may limit the collection of edge data, pushing systems toward federated or decentralized graph learning. Each of these policies shapes the hardware-aware design space.

There is also a need for interdisciplinary education. Graph neural network acceleration lies at the intersection of machine learning, computer architecture, distributed systems, and policy. Researchers trained only in kernels may overlook fairness and sustainability, while researchers trained only in governance may lack the technical vocabulary to evaluate acceleration claims. Collaborative teams and shared benchmarks can bridge this gap. The goal is not to slow innovation but to ensure that innovation is robust, accountable, and environmentally responsible.

10. Conclusion

Hardware-aware graph neural network acceleration on modern GPUs is a multidimensional challenge that cannot be reduced to a single kernel or architectural trick. Graph workloads are irregular, memory-bound, and dynamic, which makes them poorly suited to naive GPU parallelization. Effective systems must co-design graph representation, memory layout, kernel specialization, distributed partitioning, and runtime scheduling. They must also account for reproducibility, robustness, fairness, and sustainability, because these concerns are shaped by the same infrastructure decisions that determine performance.

This paper has argued that acceleration research should adopt a socio-technical perspective. The technical trade-offs among density, sparsity, sampling, fusion, precision, and communication are inseparable from institutional trade-offs among cost, access, equity, and environmental impact. As graph neural networks become embedded in recommendation, finance, healthcare, transportation, and scientific discovery, the consequences of these trade-offs will only grow. The next generation of GPU-accelerated graph systems should therefore be evaluated not only by speedup but also by their ability to support accountable, inclusive, and sustainable machine learning practice.

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. Jia, Z., Lin, S., Gao, R., Zaharia, M., & Aiken, A. (2019). Improving the accuracy, scalability, and performance of graph neural networks with roc. In *Proceedings of Machine Learning and Systems*.
8. Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.
9. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., Zhang, Z., & Li, A. (2019). Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.
10. 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 *IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms* (pp. 36–46).
11. Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., & Zaharia, M. (2019). PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (pp. 1–15).
12. Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–16).
13. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., & Krishnamurthy, A. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation* (pp. 578–594).
14. Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems* (pp. 16344–16359).
15. 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).
16. Williams, S., Waterman, A., & Patterson, D. (2009). Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4), 65–76.
17. Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L. M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*.
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).
19. Barocas, S., Hardt, M., & Narayanan, A. (2019). Fairness and machine learning: Limitations and opportunities. fairmlbook.org.

20. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model cards for model reporting. In Proceedings of the Conference on Fairness, Accountability, and Transparency (pp. 220–229).