

Accelerating Graph Transformer Models through GPU-Aware Kernel Fusion

Borys Szalas

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

Grant C. Walls

Department of Computer Science, University of Central Florida, Orlando, FL, USA.
grant.work@ucf.edu

Hardi Natsir

Department of Computer Science, George Mason University, Fairfax, VA, USA.
hardi.natsir@gmu.edu

Abstract

Graph Transformers have emerged as a powerful class of models that combine the representational flexibility of attention mechanisms with the relational structure of graphs. Their computational patterns, however, are poorly matched to conventional GPU execution strategies because they mix sparse neighborhood aggregation, dense attention, irregular memory access, and dynamic control flow. This paper examines GPU-aware kernel fusion as a system-level strategy for accelerating Graph Transformer models. Rather than treating fusion as a narrow compiler transformation, the discussion frames it as an architectural and infrastructural decision that shapes memory traffic, occupancy, synchronization, deployment economics, robustness, fairness, and sustainability. The paper first analyzes the workload characteristics of Graph Transformers and explains why unfused kernels waste bandwidth and launch overhead. It then evaluates fusion boundaries across message passing, attention, normalization, and feed-forward stages. The analysis considers trade-offs involving register pressure, shared-memory capacity, graph partitioning, batching, and distributed execution. It further situates kernel fusion within broader governance and policy contexts, including energy accounting, access equity, reproducibility, and operational resilience. Case illustrations from graph learning, large-scale GPU training, and attention optimization show that fusion can deliver substantial gains but also creates dependencies on hardware vendors, compiler maturity, and workload-specific tuning. The central argument is that GPU-aware kernel fusion is not merely an optimization technique but a socio-technical infrastructure choice whose benefits and risks must be evaluated together. The paper concludes with research directions for portable fusion, adaptive scheduling, sustainable deployment, and accountable performance reporting.

Keywords

Graph Transformers; GPU-aware kernel fusion; graph neural networks; systems for machine learning; high-performance computing; AI infrastructure governance; sustainable computing; fairness in graph learning.

1. Introduction

Graph Transformers represent a convergence of two influential lines of machine learning research. Attention-based architectures have demonstrated strong sequence modeling capabilities across language, vision, and multimodal learning [1], while graph neural networks have provided effective inductive biases for relational data [2]. Graph attention networks extended attention to neighborhood aggregation [3], and subsequent work generalized Transformer-style attention to graph-structured inputs [4]. More recent formulations have sought to combine message passing, positional encodings, structural bias, and global attention within unified architectures [5]. This convergence has produced models that can capture long-range dependencies, heterogeneous relations, and complex structural patterns. It has also produced computational graphs that are difficult to execute efficiently on modern accelerators.

The difficulty is not simply that Graph Transformers require many floating-point operations. The more serious systems challenge is that their execution is irregular. Graph neighborhoods vary in size, attention sparsity changes across layers, and memory access patterns depend on graph topology. At the same time, Transformer components such as multi-head attention and feed-forward blocks are dense and highly parallel. A Graph Transformer therefore contains two incompatible execution regimes: one that benefits from sparse gather-scatter operations and another that benefits from large, regular matrix operations. GPU-aware kernel fusion offers a way to reconcile these regimes by combining multiple operations into fewer, better-scheduled kernels.

Kernel fusion is often described as a compiler optimization that reduces memory traffic and kernel launch overhead. In practice, it is a broader systems decision. Fusion changes the granularity of parallelism, the lifetime of intermediate tensors, the pressure on shared memory and registers, and the synchronization structure of the application. It also changes how easily a model can be ported across GPU generations, cloud providers, and deployment environments. For Graph Transformers, these changes have consequences for throughput, latency, energy consumption, fault tolerance, and even fairness when access to optimized infrastructure is unevenly distributed. This paper argues that GPU-aware kernel fusion should be studied as a first-class element of Graph Transformer system design rather than as a late-stage implementation detail.

2. Architecture and Workload Characteristics of Graph Transformers

Graph Transformer architectures vary widely, but they typically combine several recurring components. These include node feature projection, graph-aware attention, message passing or neighborhood aggregation, positional or structural encoding, normalization, residual connections, and feed-forward networks. Some models emphasize local message passing with Transformer-style attention [6], while others use global attention with graph-derived biases or encodings [7]. Frameworks such as Deep Graph Library and PyTorch Geometric provide abstractions for expressing these operations [8], [9], but the underlying execution still consists of many separate GPU kernels. Sampling-based and inductive approaches further complicate the workload by introducing dynamic neighborhoods during training [10]. Graphormer-style models show that attention can be effective for graph representation learning when structural information is injected appropriately [11]. The result is a workload that is neither purely sparse nor purely dense.

The first major characteristic is irregularity. In a graph attention layer, each node attends to a subset of neighbors or to all nodes with a structural mask. The number of valid interactions can vary by orders of magnitude across nodes. GPU threads assigned to different nodes may therefore perform different amounts of work, causing divergence and underutilization. When

attention is global, the computation becomes denser, but the structural mask still introduces irregularity in memory access and arithmetic. The second characteristic is mixed precision and mixed data locality. Node features may be small enough to fit in cache for some partitions but not others. Edge lists, adjacency structures, and attention masks compete for bandwidth with intermediate activations. The third characteristic is layer heterogeneity. Early layers may be dominated by local aggregation, while later layers may spend more time on global attention and feed-forward computation.

These characteristics make naive kernel-per-operation execution expensive. A single Graph Transformer layer can launch dozens of kernels, each reading and writing large intermediate tensors. The overhead is not only launch latency but also lost opportunity for data reuse. If normalization, residual addition, and feed-forward projection are executed separately, the intermediate values must travel to global memory and back. If attention scores are materialized before softmax and dropout, memory traffic increases sharply. If neighborhood aggregation is separated from attention, graph structure data may be reread multiple times. These inefficiencies are especially damaging because Graph Transformers often operate at the boundary of memory capacity and compute utilization. Reducing data movement is therefore as important as reducing arithmetic.

3. GPU Execution Constraints and Kernel Fusion Opportunities

Modern GPUs provide massive parallelism, but they also impose strict constraints on memory bandwidth, shared-memory capacity, register usage, and synchronization. Large-scale training systems have shown that partitioning, communication overlap, and memory optimization are essential for scale [12], [14]. Compiler frameworks such as TVM and LLVM provide mechanisms for generating and transforming kernels [15], [16], but achieving high performance still requires hardware-aware scheduling. Studies of Transformer optimization have found that data movement often dominates runtime, even when matrix multiplication is highly optimized [17]. FlashAttention demonstrated that IO-aware attention can reduce memory traffic and improve efficiency by restructuring the computation around on-chip memory [18]. Serving systems such as PagedAttention show similar lessons for inference, where memory management determines throughput and latency [19]. These findings directly inform Graph Transformer fusion.

The most obvious fusion opportunity is the attention pipeline. In a conventional implementation, query, key, and value projections are separate kernels; attention scores are written to global memory; softmax reads and writes them; dropout may add another pass; and the output projection reads the attention result. Fusion can combine projection, score computation, masking, softmax, and value aggregation into a smaller number of kernels. For graphs, masking is especially important because structural masks, padding masks, and edge-type biases may all apply. Fusing mask application with score computation avoids materializing large masked tensors. Fusing softmax with value aggregation reduces intermediate storage. However, this fusion increases register pressure and may limit occupancy, particularly when attention heads and hidden dimensions are large.

A second opportunity lies in message passing and neighborhood aggregation. Graph neural network kernels often suffer from irregular memory access and poor cache reuse. GPU-aware fusion can combine gather, feature transformation, edge conditioning, and aggregation. One representative effort [13] demonstrates that extremely fast GPU kernel design for graph neural network training can be achieved by rethinking memory access, workload balancing, and kernel structure. In Graph Transformers, similar principles apply, but the fusion boundary

must also respect attention. If message passing is fused too aggressively with attention, the kernel may become too complex for the compiler to schedule effectively. If it is not fused enough, edge data and node features are repeatedly fetched from global memory. The optimal boundary depends on graph density, feature dimension, batch size, and GPU architecture.

A third opportunity involves normalization, residual connections, and feed-forward layers. These operations are elementwise or reduction-based and often follow attention. Fusing normalization with residual addition reduces global memory round trips. Fusing the first feed-forward projection with activation can improve locality, especially when the hidden dimension is large. However, layer normalization requires reductions across features, which introduces synchronization within a thread block or across blocks. Fusing reductions with other operations can create barriers that reduce parallelism. The trade-off is therefore not simply fewer kernels but better kernel shapes. A fused kernel must have enough independent work to hide memory latency, enough registers to avoid spills, and enough shared memory to stage graph or attention data without exceeding hardware limits.

4. System-Level Design Trade-offs in GPU-Aware Fusion

The design space of GPU-aware kernel fusion for Graph Transformers is shaped by several structural trade-offs. The first is between memory savings and occupancy. Fusion reduces intermediate tensor traffic, but it also increases the state held within a kernel. More live variables mean more registers, and high register usage can reduce the number of active warps. For graph attention, holding query blocks, key blocks, value blocks, edge masks, and accumulators simultaneously may exceed register capacity. Shared memory can absorb some of this pressure, but shared memory is also limited and bank conflicts can degrade performance. The result is a tension between reducing global memory traffic and maintaining enough parallelism to saturate the GPU.

The second trade-off is between generality and specialization. A fused kernel specialized for a particular graph density, feature dimension, and attention pattern can achieve high performance, but it may not generalize across models. Systems such as TVM and LLVM offer intermediate representations for transformation [15], [16], but automatic fusion for irregular graph workloads remains difficult. A specialized kernel may also depend on vendor-specific features, limiting portability across AMD, Intel, and emerging accelerators. In contrast, a general fused kernel may be easier to maintain but may leave substantial performance on the table. For production systems, this trade-off interacts with deployment constraints. Cloud environments may present heterogeneous GPU generations, and a model that is optimized for one architecture may underperform on another.

The third trade-off is between static and dynamic scheduling. Static fusion plans are determined at compile time, while dynamic plans adapt to runtime graph properties. Graph Transformers often have variable neighborhood sizes and changing sparsity across layers. A static plan may work well for one batch but poorly for another. Dynamic fusion can select different kernel boundaries based on graph statistics, memory availability, and current occupancy. However, dynamic scheduling introduces runtime overhead and complicates reproducibility. It also makes performance less predictable, which is problematic for service-level agreements. The fourth trade-off is between local and distributed execution. Fusing kernels within a single GPU reduces communication with global memory, but distributed training requires exchanging gradients, activations, or embeddings across devices. Fusion can reduce the volume of intermediate data that must be communicated if it keeps more computation local, but it can also complicate gradient accumulation and checkpointing.

A particularly important structural consideration is the interaction between fusion and graph partitioning. Graph Transformers often rely on sampling or clustering to create mini-batches. If partitions are poorly balanced, some GPU streaming multiprocessors receive much more work than others. Fusion can exacerbate this imbalance if it creates larger, longer-running kernels that cannot be load-balanced as easily as many small kernels. Conversely, fusion can improve balance if it combines operations that have complementary resource needs. For example, a memory-bound aggregation step can be fused with a compute-bound projection step so that different execution units are utilized concurrently. Achieving this balance requires coordination between graph partitioning, kernel scheduling, and memory management.

5. Infrastructure, Deployment, and Operational Governance

GPU-aware kernel fusion is not deployed in a vacuum. It depends on compiler toolchains, GPU drivers, libraries, container images, orchestration systems, and monitoring infrastructure. Large-scale training frameworks rely on parallelism strategies, memory optimizations, and communication overlap [12], [14]. Inference serving systems depend on memory management and batching policies [19]. Compiler frameworks such as TVM and LLVM shape what optimizations are feasible without hand-written kernels [15], [16]. This infrastructure creates path dependencies. Organizations that invest in a particular vendor ecosystem may gain performance advantages but also become locked into specific hardware and software stacks. For public institutions, smaller research groups, and organizations in resource-constrained regions, these dependencies can translate into unequal access to efficient Graph Transformer deployment.

Operational governance must therefore address more than raw speed. It must consider reproducibility, observability, and accountability. A fused kernel may be difficult to inspect because intermediate tensors are not materialized. Debugging numerical errors becomes harder when many operations are combined. Performance claims may depend on specific driver versions, compiler flags, and GPU models. Without disciplined reporting, results may not be reproducible. Governance frameworks for algorithmic systems emphasize transparency, explainability, and accountability [23], [24]. In the context of kernel fusion, these principles imply that performance reports should include hardware configuration, software versions, graph characteristics, and fusion strategies. They also imply that optimization decisions should be documented when they affect model behavior, latency, or energy consumption.

Deployment governance also involves resilience. Fused kernels may be more sensitive to out-of-memory conditions, because they hold more intermediate state. They may be more sensitive to hardware faults, because a single long-running kernel can fail after substantial computation. They may be less portable, because they exploit architecture-specific features. Operational teams must therefore balance performance with fault tolerance. Techniques such as checkpointing, fallback kernels, and adaptive batching can improve resilience. However, these measures add complexity. A system that switches between fused and unfused paths must ensure numerical consistency and predictable latency. In regulated domains such as healthcare, finance, and public services, such consistency can be as important as peak throughput.

6. Robustness, Fairness, and Policy Implications

The fairness implications of GPU-aware kernel fusion may seem indirect, but they are significant. Graph Transformers are increasingly used for recommendation, fraud detection, social network analysis, transportation, and scientific discovery. The efficiency of these

systems affects who can train and deploy them. Organizations with access to advanced GPUs and mature fusion compilers can iterate faster and serve larger models at lower cost. Organizations without such access may be relegated to smaller models, longer training times, or third-party APIs. This uneven capability can shape the kinds of problems that are addressed and the populations that are served. Ethical analyses of algorithms have long warned that technical choices can encode social values and distribute benefits unevenly [23]. Sustainable AI research similarly emphasizes that efficiency gains do not automatically translate into equitable outcomes [22].

Robustness is another concern. Fusion can improve performance by reducing memory traffic, but it can also alter numerical behavior. Reordering operations, combining reductions, and changing precision can introduce small differences that propagate through training. In graph learning, these differences may interact with sampling, neighborhood aggregation, and attention masks. If a model is deployed in a high-stakes setting, small variations in predictions can have meaningful consequences. Fairness metrics may also shift if optimization changes which nodes are processed together or which neighborhoods are sampled. A system that is faster but less stable across demographic groups or graph communities may be unacceptable. Policy frameworks for trustworthy AI call for robustness, privacy, and non-discrimination [24]. Kernel fusion should therefore be evaluated not only with throughput benchmarks but also with stability, error analysis, and subgroup performance.

Policy implications extend to energy and procurement. Governments and institutions that fund AI infrastructure make decisions about GPU clusters, cloud contracts, and data centers. If fusion reduces energy per inference or per training epoch, it can contribute to sustainability goals. However, efficiency gains can also lead to rebound effects, where lower cost encourages more usage and total energy increases. Research on green AI and sustainable computing has highlighted the need for careful measurement and lifecycle analysis [21], [22]. Public policy should require transparent reporting of energy use, hardware utilization, and optimization strategies. It should also support open compiler infrastructure and portable fusion techniques so that efficiency benefits are not concentrated among a few vendors. Without such policies, kernel fusion may deepen existing inequalities in computational research and deployment.

7. Sustainability and Economic Considerations

The sustainability case for GPU-aware kernel fusion is straightforward at the level of individual operations. Fewer kernels mean fewer launches, less memory traffic, and often lower energy consumption. Studies of deep learning energy use have shown that training and inference can have substantial carbon footprints [20]. Green AI research argues for efficiency metrics that include energy and carbon, not only accuracy or speed [21]. Sustainable AI frameworks call for system-level optimization across hardware, software, and data center operations [22]. Fused Graph Transformer kernels can contribute to these goals by improving utilization and reducing wasted computation. However, the actual environmental benefit depends on how the GPU is powered, how long it remains idle, and whether the efficiency gain increases total demand.

Economic considerations are equally important. GPU time is expensive, and Graph Transformers are often trained on large graphs that require many epochs. Fusion can reduce training cost and make real-time inference more feasible. This can enable new applications in transportation, logistics, healthcare, and scientific simulation. It can also concentrate market power. Firms that master hardware-aware fusion can offer lower prices and better

performance, making it harder for smaller competitors to enter. Open-source compilers and portable kernel libraries can mitigate this concentration, but they require sustained investment. The economics of fusion therefore involve a balance between private incentive and public infrastructure. Policies that fund shared GPU facilities, open benchmarks, and reproducible optimization research can help distribute the benefits more broadly.

8. Case Illustrations and Cross-Domain Comparisons

Graph learning provides several instructive cases. Classical graph convolutional networks rely on neighborhood aggregation and are relatively sparse [2]. Graph Transformers add attention, which introduces dense matrix operations and larger intermediate tensors. Scalable graph Transformer designs attempt to retain expressiveness while controlling complexity [7]. GPU kernel designs for graph neural network training show that memory access and workload balancing are central to performance [13]. FlashAttention shows that IO-aware attention can dramatically improve efficiency by restructuring computation around on-chip memory [18]. Serving systems show that memory management and batching determine practical throughput [19]. Together, these cases suggest that Graph Transformer acceleration requires a fusion strategy that spans sparse and dense phases rather than optimizing each phase in isolation.

Cross-domain comparisons reveal similar patterns. In natural language processing, Transformer optimization has focused on attention fusion, memory-efficient attention, and distributed training [12], [17], [18]. In recommendation and search, graph-based retrieval systems combine sparse traversal with dense ranking models. In scientific computing, graph transformers may be used for molecular property prediction, materials discovery, and climate modeling, where irregular mesh structures resemble graph topologies. Each domain has different latency, throughput, and accuracy requirements. A fusion strategy that works for large-scale offline training may not work for latency-sensitive inference. A strategy that works for homogeneous graphs may not work for heterogeneous or dynamic graphs. The lesson is that GPU-aware fusion must be co-designed with the application domain, not treated as a universal switch.

9. Future Directions

Several research directions follow from this analysis. First, portable fusion compilers should be extended to handle irregular graph workloads. Current compiler frameworks provide strong foundations [15], [16], but they need better abstractions for sparse gathers, dynamic masks, and graph partitioning. Second, adaptive fusion should be developed for runtime conditions. A Graph Transformer could select fusion boundaries based on graph density, feature dimension, batch size, and GPU occupancy. Third, fusion should be integrated with distributed training and serving. Large-scale systems already rely on memory optimization and communication overlap [12], [14], and serving systems rely on efficient memory management [19]. Fusion should complement these mechanisms rather than conflict with them.

Fourth, evaluation should become more holistic. Performance papers should report energy, fairness, robustness, and reproducibility alongside throughput. This aligns with responsible AI frameworks [23], [24] and sustainability research [20], [21], [22]. Fifth, governance should support open benchmarks and shared infrastructure. If fusion techniques remain proprietary, the benefits may accrue mainly to large firms. Public investment in open compilers, shared GPU clusters, and standardized reporting can broaden access. Finally, Graph Transformer architectures themselves may evolve to be more fusion-friendly. Models could expose

structured sparsity, block-wise attention, or predictable neighborhood sizes [5], [7]. Architecture and system design should therefore proceed together, with fusion in mind from the beginning.

10. Conclusion

GPU-aware kernel fusion is a powerful approach to accelerating Graph Transformer models, but it is not a simple optimization. It changes memory behavior, parallelism, portability, energy use, and operational risk. This paper has argued that fusion should be understood as a system-level and socio-technical design choice. The workload characteristics of Graph Transformers combine sparse graph irregularity with dense attention computation, making fusion both valuable and difficult. Attention pipelines, message passing, normalization, and feed-forward layers offer opportunities for fusion, but each opportunity involves trade-offs in register pressure, shared memory, occupancy, generality, and resilience. Infrastructure and governance shape who can benefit from these optimizations, while fairness, robustness, and sustainability require broader evaluation criteria. The most promising path forward is co-design: Graph Transformer architectures, GPU kernels, compiler toolchains, deployment systems, and policy frameworks should be developed together. Only then can fusion deliver performance gains that are not only fast but also portable, accountable, and broadly beneficial.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 5998–6008). Curran Associates.
2. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.
3. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.
4. Yun, S., Jeong, M., Kim, R., Kang, J., & Kim, H. J. (2019). Graph transformer networks. In *Advances in Neural Information Processing Systems* (Vol. 32, pp. 11983–11993).
5. Dwivedi, V. P., & Bresson, X. (2020). A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*.
6. Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., & Sun, Y. (2021). Masked label prediction: Unified message passing model for semi-supervised classification. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence* (pp. 1548–1554). IJCAI.
7. Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., & Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems* (Vol. 35, pp. 14501–14515).
8. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, Z., Guo, J., Li, H., & Zhang, Z. (2019). Deep Graph Library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*.
9. Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

10. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 1024–1034).
11. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., & Liu, T. Y. (2021). Do Transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems* (Vol. 34, pp. 28877–28888).
12. Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., & Zaharia, M. (2021). Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–15). ACM.
13. 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).
14. Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). ZeRO: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–16). IEEE.
15. 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 *13th USENIX Symposium on Operating Systems Design and Implementation* (pp. 578–594).
16. Lattner, C., & Adve, V. (2004). LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization* (pp. 75–88). IEEE.
17. Ivanov, A., Dryden, N., Ben-Nun, T., Li, S., & Hoefler, T. (2021). Data movement is all you need: A case study on optimizing transformers. In *Proceedings of Machine Learning and Systems* (Vol. 3, pp. 711–732).
18. 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* (Vol. 35, pp. 16344–16359).
19. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 611–626). ACM.
20. 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). ACL.
21. Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12), 54–63.
22. Wu, C. J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., Gschwind, M., Gupta, A., Ott, M., Melnikov, A., Candido, S.,

- Brooks, D., Chauhan, G., Lee, B., Lee, H. H., ... & Hazelwood, K. (2022). Sustainable AI: Environmental implications, challenges and opportunities. In *Proceedings of Machine Learning and Systems* (Vol. 4, pp. 795–813).
23. Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S., & Floridi, L. (2016). The ethics of algorithms: Mapping the debate. *Big Data & Society*, 3(2), 1–21.
24. Floridi, L., Cows, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., Luetge, C., Madelin, R., Pagallo, U., Rossi, F., Schafer, B., Valcke, P., & Vayena, E. (2018). AI4People—An ethical framework for a good AI society: Opportunities, risks, principles, and recommendations. *Minds and Machines*, 28(4), 689–707.
25. Benjamin, R. (2019). Race after technology: Abolitionist tools for the new Jim code. *Polity*.