

Low-Memory Graph Neural Network Training through Kernel-Level Optimization

Ciro Mondaini

Department of Computer Science, George Mason University, Fairfax, VA, USA.
mondaini1996@gmu.edu

Tellervo Vartiainen

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

Marissa Ali

Department of Computer Science and Engineering, University of Nevada, Reno, NV, USA.
alimarissa@unr.edu

Abstract

Graph neural networks have become a central computational method for learning over relational data, but their training remains constrained by severe memory pressure, irregular data access, and expensive message-passing operations. This paper examines low-memory graph neural network training through the lens of kernel-level optimization, arguing that meaningful memory reduction cannot be achieved solely through model compression or graph sampling. Instead, it requires coordinated redesign of execution kernels, memory hierarchies, scheduling policies, and data movement across heterogeneous hardware. The analysis situates kernel-level optimization within broader system architecture, infrastructure, governance, sustainability, robustness, and fairness debates. It considers how fused kernels, sparse memory layouts, recomputation, quantization, unified memory, and distributed partitioning alter the trade-offs among throughput, accuracy, energy use, and deployability. The paper also explores the institutional consequences of memory-efficient graph learning, including the possibility of more inclusive access to graph artificial intelligence, the risk of new forms of technical lock-in, and the need for governance frameworks that treat efficiency as a socio-technical property rather than a purely computational metric. It concludes that kernel-level memory optimization is not merely an implementation detail but a foundational condition for scalable, equitable, and environmentally responsible graph learning systems. The discussion is intended for researchers and practitioners who design graph learning platforms, hardware accelerators, distributed training infrastructures, and policy frameworks for responsible artificial intelligence.

Keywords

graph neural networks; kernel-level optimization; memory hierarchy; GPU training; system architecture; sustainable computing; robustness; fairness; governance.

1. Introduction

Graph neural networks have emerged as a powerful family of machine learning models for domains in which entities and relations cannot be reduced to independent samples. Social networks, molecular structures, transportation systems, recommendation platforms, power

grids, and scientific knowledge graphs all exhibit relational dependencies that conventional vector-based models represent poorly. Early formulations of graph neural networks established recursive message passing as a general mechanism for learning over graph-structured data [1]. Subsequent work on graph convolutional networks simplified spectral-inspired propagation into practical neighborhood aggregation [2], while inductive methods enabled generalization to unseen nodes and graphs [3]. Attention-based graph models further increased expressive capacity by learning adaptive neighbor weights [4]. Surveys of the field have documented rapid methodological expansion across supervised, unsupervised, and self-supervised settings [5], and application-oriented reviews have traced adoption in chemistry, biology, medicine, transportation, and recommender systems [6].

Despite this progress, graph neural network training remains disproportionately expensive relative to many other deep learning workloads. The primary difficulty is not only the number of parameters but the irregularity of graph computation. Neighborhood sizes vary widely, feature dimensions can be large, and message passing often requires gathering, scattering, and reducing data across sparse index structures. These operations stress memory bandwidth, cache hierarchies, and interconnect fabric. As graphs grow to billions of edges, training pipelines become constrained by the cost of moving data rather than the cost of arithmetic. Hardware accelerators such as graphics processing units have improved throughput, but their programming models still struggle with graph workloads because sparse access patterns create poor locality and load imbalance. Surveys of graph neural network acceleration have therefore emphasized the need for co-design across algorithms, compilers, runtime systems, and hardware [7].

This paper focuses on low-memory graph neural network training through kernel-level optimization. The central claim is that memory efficiency is not a secondary concern to be addressed after model design. It is a first-order architectural property that shapes which graphs can be trained, where training can occur, how quickly models can be updated, and who can participate in graph artificial intelligence development. Kernel-level optimization refers to the design of low-level execution routines that implement neighborhood aggregation, feature transformation, gradient computation, and parameter updates. These routines determine how data is staged in registers, shared memory, caches, and higher-bandwidth memory. They also influence whether intermediate activations are stored, recomputed, compressed, or discarded. Recent work on extremely fast graphical processing unit kernel design demonstrates that aggressive kernel specialization can substantially improve training speed and memory behavior for graph neural networks [8]. Yet such optimization must be understood within a wider system context.

The argument proceeds through several stages. First, the paper reviews the structural sources of memory pressure in graph neural network training. Second, it examines kernel-level mechanisms such as fusion, tiling, sparse layouts, recomputation, quantization, and unified memory. Third, it analyzes trade-offs involving scalability, robustness, infrastructure, and deployment. Fourth, it connects these technical choices to fairness, governance, sustainability, and policy. The overall aim is to show that low-memory training is a socio-technical objective. It requires not only faster kernels but also transparent benchmarks, reproducible systems, equitable access to hardware, and governance mechanisms that recognize the environmental and distributive consequences of efficiency decisions.

2. Background and Problem Setting

Graph neural networks operate by iteratively updating node representations through message passing. In each layer, a node collects information from its neighbors, transforms the aggregated information, and combines it with its own state. This design is conceptually elegant, but its computational behavior is highly irregular. Real-world graphs often follow power-law degree distributions, meaning that a small number of nodes have enormous neighborhoods while most nodes have few connections. Dense matrix operations cannot efficiently represent this skew. Sparse operations reduce arithmetic waste but introduce indirect memory access, poor cache reuse, and synchronization overhead. As a result, graph neural network training is frequently memory-bound rather than compute-bound. The cost of fetching neighbor features, intermediate activations, and gradients can dominate the cost of floating-point operations.

The memory problem is compounded by depth and batching. Deeper graph neural networks require storing activations for backward propagation. Because neighborhood expansion grows rapidly with depth, naive full-batch training can exceed device memory even for moderate graphs. Sampling methods address this issue by selecting subsets of neighbors or subgraphs during training [3]. However, sampling introduces variance, changes convergence behavior, and may weaken performance on tasks that require long-range dependencies. Graph partitioning approaches distribute the graph across devices, but they create communication overhead and load imbalance. Systems such as AliGraph illustrate the platform-level complexity of managing graph storage, sampling, and neural network training in an integrated environment [11]. Frameworks such as the Deep Graph Library and PyTorch Geometric provide abstractions that simplify implementation, but they also mediate how memory is allocated and reused [9], [10]. Therefore, kernel-level behavior cannot be separated from framework design.

Another source of pressure is feature dimensionality. Many graph learning tasks involve rich node attributes, such as text embeddings, molecular descriptors, or multimodal sensor readings. Storing features for all nodes can consume enormous memory, especially when the graph is large and the feature dimension is high. Gradient storage adds further cost. Optimizers may maintain momentum and variance estimates, multiplying the memory footprint of trainable parameters. In distributed settings, replicas, gradients, and communication buffers add additional layers of consumption. Research on memory-efficient deep learning has produced techniques such as sublinear memory training through recomputation [12] and dynamic tensor rematerialization [13]. These methods trade computation for memory by discarding intermediate tensors and reconstructing them when needed. Zero redundancy optimizers and model-parallel training strategies have similarly reduced memory pressure in large-scale language and vision models [14], [15]. Graph neural networks can benefit from these ideas, but their irregular structure makes direct transfer difficult.

The evaluation of low-memory graph learning is also complicated by dataset and benchmark design. The Open Graph Benchmark provides standardized datasets and evaluation protocols that improve comparability across methods [16]. Nevertheless, benchmarks often emphasize accuracy and throughput on a limited set of hardware configurations. They may not report peak memory, energy consumption, or sensitivity to partitioning. This omission is consequential because a method that reduces memory on one accelerator may increase communication or recomputation costs on another. It may also shift the burden from device memory to host memory or storage, creating hidden infrastructure dependencies. A rigorous

account of low-memory training must therefore consider the entire memory hierarchy and the operational context in which training occurs.

3. Kernel-Level Memory Optimization: Architecture and Mechanisms

Kernel-level optimization for graph neural networks begins with the recognition that memory movement is the dominant cost. A kernel is the executable unit that performs a specific operation on a device. In graph learning, kernels handle operations such as neighbor gathering, feature transformation, aggregation, activation, loss computation, and gradient reduction. When these operations are implemented as separate kernels, intermediate results must be written to global memory and read back. This round-trip consumes bandwidth and increases latency. Kernel fusion combines multiple operations into a single execution pass, allowing intermediate values to remain in registers or shared memory. Fusion can significantly reduce memory traffic, but it increases register pressure and may reduce occupancy. The design challenge is to fuse enough operations to reduce memory movement without exceeding the resources available to each thread block.

Tiling and blocking are related strategies. By processing graph neighborhoods in tiles that fit within fast on-chip memory, a kernel can reuse feature data across multiple edges. However, graph irregularity complicates tiling because neighbor sets are not rectangular. Some systems sort edges by source or destination to improve locality, while others use compressed sparse row, compressed sparse column, or coordinate formats. Each format has different implications for memory overhead, parallel scan efficiency, and load balance. Recent high-performance kernel designs have explored maximum kernel strategies that reorganize computation to maximize data reuse and minimize redundant memory access during training [8]. Such work demonstrates that low-level decisions about thread mapping, memory coalescing, and instruction scheduling can produce substantial gains. Yet these gains are not automatic. They depend on graph properties, feature dimensions, batch sizes, and hardware generations.

Recomputation is another central mechanism. Instead of storing all intermediate activations during the forward pass, a training system can store only selected checkpoints and recompute the rest during backpropagation. This approach reduces activation memory at the cost of additional computation. Sublinear memory training and dynamic tensor rematerialization provide general frameworks for this trade-off [12], [13]. In graph neural networks, recomputation must account for neighborhood sampling and sparse index structures. Recomputing a node embedding may require revisiting neighbors that were sampled in the forward pass. If the sampling pattern is not preserved, recomputation may produce inconsistent gradients. Therefore, memory-efficient graph training requires careful coordination between sampling logic, kernel execution, and gradient computation. The system must decide which activations are cheap to recompute, which are expensive, and which are essential to retain.

Quantization and mixed-precision arithmetic can also reduce memory. Representing features, weights, and gradients in lower precision decreases storage and bandwidth requirements. However, graph neural networks are sensitive to numerical precision because aggregation can involve many small contributions. Aggressive quantization may degrade accuracy, especially for tasks with heterogeneous feature scales or long-range dependencies. Kernel-level support for mixed precision must therefore include scaling, accumulation in higher precision, and careful handling of sparse reductions. Unified memory and page migration offer another avenue by allowing data to reside in host memory and migrate to device memory on demand. This can lower peak device memory usage, but it may introduce page faults and unpredictable

latency. For large graphs, unified memory can be effective when access patterns are predictable, but irregular graph traversal may cause thrashing. Kernel designers must therefore align memory placement policies with graph partitioning and traversal order.

The architecture of the accelerator itself shapes what is possible. Graphics processing units provide high bandwidth memory, large numbers of parallel threads, and hierarchical caches. However, their performance depends on coalesced access and sufficient parallelism. Graph kernels often suffer from divergence because neighboring nodes have different degrees and feature lengths. Specialized accelerators for graph processing may offer different trade-offs, such as larger on-chip memory, reconfigurable interconnects, or sparse arithmetic units. Surveys of graph neural network acceleration have categorized these approaches and highlighted the need for hardware-software co-design [7]. In practice, no single kernel-level optimization dominates all scenarios. The optimal design depends on whether the workload is training or inference, whether the graph is static or dynamic, whether features are dense or sparse, and whether the deployment target is a data center, an edge device, or a federated environment.

4. System Trade-Offs and Infrastructure Implications

Low-memory graph neural network training creates a series of system trade-offs that extend beyond a single kernel. The first trade-off is between memory and computation. Recomputation reduces memory but increases arithmetic and memory access elsewhere. Fusion reduces global memory traffic but may lower occupancy and increase register spills. Quantization reduces storage but can harm convergence and require additional calibration. These trade-offs are not independent. A recomputation strategy may become more attractive when kernels are fused and can recompute intermediate values efficiently. A quantization strategy may become less attractive when sparse access patterns already dominate latency. System designers must therefore evaluate combinations of techniques rather than treating them as isolated optimizations.

The second trade-off is between scale and communication. Distributed training divides a graph across multiple devices or nodes. Partitioning can reduce per-device memory, but it introduces cross-device messages for edges that span partitions. If the graph has poor community structure or high edge cut, communication can overwhelm the savings. Model-parallel and data-parallel strategies developed for large neural networks provide useful lessons [14], [15]. However, graph learning adds the complication that the data itself is relational. Splitting nodes and edges changes the semantics of neighborhood aggregation unless the system carefully manages halo regions and remote dependencies. Infrastructure choices such as interconnect bandwidth, topology, and storage latency therefore become part of the memory optimization problem. A kernel that is efficient on a single device may be inefficient in a distributed setting if it generates excessive remote memory traffic.

The third trade-off is between flexibility and specialization. General-purpose graph learning frameworks support many model architectures, sampling strategies, and hardware backends [9], [10]. Specialized kernels can achieve better memory behavior but may only support specific operations or graph formats. This tension affects maintainability, portability, and long-term sustainability. A research group may achieve excellent results with a custom kernel, but reproducing or extending that kernel may require specialized expertise. Platform-level systems such as AliGraph demonstrate the value of integrated graph storage, sampling, and training, but they also illustrate the complexity of maintaining such a platform [11]. The

governance of graph learning infrastructure must therefore consider not only raw performance but also the ability of diverse institutions to audit, adapt, and operate the technology.

The fourth trade-off concerns robustness. Memory optimization can introduce new failure modes. Recomputation may produce nondeterministic gradients if sampling is not controlled. Quantization may cause silent accuracy loss on underrepresented subgraphs. Unified memory may cause unpredictable latency spikes under memory pressure. Distributed partitioning may create stragglers when some partitions have much higher degree or feature density. These failure modes are especially concerning in high-stakes domains such as healthcare, finance, and public infrastructure. Robustness requires instrumentation that measures not only average throughput but tail latency, memory fragmentation, and sensitivity to graph perturbations. It also requires fallback mechanisms that can degrade gracefully when memory is exhausted. A system that achieves low average memory usage but fails unpredictably on certain graph regions is not robust enough for production deployment.

5. Robustness, Fairness, and Governance

The relationship between memory optimization and fairness is not immediately obvious, but it is significant. Graph neural networks are often used to make decisions about people, organizations, and communities. Social recommendations, credit scoring, fraud detection, and public health forecasting can all be framed as graph learning problems. When memory constraints force the use of sampling, quantization, or partitioning, the resulting approximations may affect different parts of the graph unevenly. For example, low-degree nodes may be sampled less frequently, while high-degree hubs dominate training. Underrepresented communities may form weak or fragmented subgraphs that are poorly served by aggressive memory reduction. Bias and fairness research has shown that machine learning systems can amplify historical inequities through data, model design, and deployment choices [17]. Memory optimization adds another layer: it determines which relational patterns are visible to the model and which are compressed away.

Governance frameworks must therefore treat kernel-level decisions as consequential. Transparency about sampling rates, precision levels, partition boundaries, and recomputation policies is necessary for auditing. If a model’s training pipeline uses aggressive quantization to fit within device memory, evaluators need to know whether accuracy disparities exist across demographic or geographic subgroups. If a distributed system partitions a graph in a way that isolates certain communities, fairness audits must examine cross-partition effects. Ethical and legal debates about algorithmic accountability have emphasized the importance of explainability, contestability, and oversight [21], [22]. These principles apply to graph learning infrastructure as well. A low-memory training system is not neutral simply because it is efficient. Its efficiency is achieved through specific design choices that allocate computational resources and approximations unevenly.

There is also a risk of technical lock-in. If low-memory training depends on proprietary kernels, specialized accelerators, or closed-source libraries, then access to graph artificial intelligence may become concentrated among organizations that can afford the required infrastructure. Open frameworks and benchmarks help mitigate this risk [9], [10], [16]. However, openness at the framework level does not guarantee openness at the kernel level. Many high-performance kernels are optimized for particular hardware and may not be portable. Policy interventions could encourage public benchmarking of memory, energy, and fairness metrics. They could also support shared computational infrastructure for researchers who cannot afford large-scale accelerators. Federated learning offers one model for

distributing training across institutions while keeping data local [20]. Yet federated graph learning faces additional challenges because graph edges may cross institutional boundaries. Memory optimization in this context must address not only device constraints but also privacy, communication, and trust.

6. Deployment, Sustainability, and Policy

Deployment context strongly influences which memory optimizations are appropriate. In a data center, abundant memory and high-bandwidth interconnects may make recomputation less attractive and distributed partitioning more feasible. In an edge environment, memory is severely limited, energy is constrained, and connectivity may be intermittent. A low-memory kernel designed for a server-class graphics processing unit may not translate to a mobile accelerator or an embedded system. This divergence has implications for equity. If graph learning applications are optimized only for high-end hardware, then communities and organizations with modest infrastructure may be excluded from using or benefiting from the technology. Sustainable deployment requires a portfolio of optimization strategies that can scale down as well as up.

Energy consumption is another critical dimension. Training large graph neural networks requires electricity for computation, cooling, and data movement. The environmental cost of artificial intelligence has received increasing attention, with calls for measuring and reducing the carbon footprint of machine learning workloads [18], [19]. Kernel-level memory optimization can contribute to sustainability by reducing data movement, which is often more energy-intensive than arithmetic. However, efficiency gains can be offset by rebound effects. If training becomes cheaper, organizations may train larger models on larger graphs more frequently, increasing total energy use. Policy frameworks should therefore encourage absolute reductions in energy and memory consumption, not merely improvements in per-operation efficiency. Reporting standards could require disclosure of memory footprints, hardware utilization, and energy sources for large graph learning experiments.

Infrastructure policy also matters. Government funding agencies, universities, and industry labs shape which research directions are pursued. If funding prioritizes raw speed on standard benchmarks, then low-memory techniques that improve access or fairness may be undervalued. If procurement policies favor proprietary accelerators, then open kernel development may stagnate. A more balanced approach would support reproducible systems research, open-source kernel libraries, and interdisciplinary teams that include computer architects, machine learning researchers, social scientists, and policy experts. Governance of artificial intelligence increasingly emphasizes risk management, accountability, and human rights [22]. Graph learning systems should be included in these discussions because they model relationships that can reveal sensitive information or reinforce social stratification. Memory-efficient training does not eliminate these risks, but it can make auditing and distributed oversight more feasible if designed with transparency in mind.

7. Conclusion

Low-memory graph neural network training through kernel-level optimization is a systems problem with far-reaching implications. The irregular structure of graph computation makes memory bandwidth, cache behavior, and intermediate storage central constraints. Kernel fusion, sparse memory layouts, recomputation, quantization, unified memory, and distributed partitioning offer powerful mechanisms for reducing memory pressure, but each introduces trade-offs involving computation, communication, accuracy, robustness, and portability.

Recent work on high-performance kernel design shows that substantial gains are possible when low-level execution is co-designed with graph structure and hardware characteristics [8]. Yet these gains must be evaluated within a broader socio-technical framework that includes infrastructure access, environmental sustainability, fairness, and governance.

The paper has argued that memory optimization is not a neutral technical exercise. It determines which graphs can be trained, which organizations can train them, and which communities are represented in the resulting models. It affects energy consumption and therefore environmental responsibility. It shapes the robustness of deployed systems and the feasibility of auditing them. A responsible research agenda should therefore combine kernel-level innovation with open benchmarks, transparent reporting, and inclusive infrastructure policy. Future work should investigate adaptive memory management that responds to graph properties at runtime, cross-layer co-design between samplers and kernels, and evaluation protocols that measure fairness and sustainability alongside speed and accuracy. Without such integration, low-memory graph learning risks becoming a narrow optimization for well-resourced actors rather than a foundation for equitable and resilient relational artificial intelligence.

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.
2. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations* (pp. 1-14).
3. 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).
4. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations* (pp. 1-12).
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. 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.
8. 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).
9. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., & Zhang, Z. (2019). Deep graph library: A graph-

centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315.

10. Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. arXiv preprint arXiv:1903.02428.
11. Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y., & Zhou, J. (2019). AliGraph: A comprehensive graph neural network platform. *Proceedings of the VLDB Endowment*, 12(12), 2094-2105.
12. Chen, T., Xu, B., Zhang, C., & Guestrin, C. (2018). Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174.
13. Kirisame, M., Zhang, S. L., Khandelwal, D., Sen, K., & Matsuoka, S. (2021). Dynamic tensor rematerialization. In *Proceedings of the 9th International Conference on Learning Representations* (pp. 1-19).
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).
15. 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).
16. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., & Leskovec, J. (2020). Open Graph Benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 22118-22133).
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.
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. Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12), 54-63.
20. Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., ... & Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2), 1-210.
21. 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.
22. Cath, C. (2018). Governing artificial intelligence: Ethical, legal and technical opportunities and challenges. *Philosophical Transactions of the Royal Society A*, 376(2133), 20180080.