

Hardware-Aware Graph Neural Network Optimization for Scalable Integrated Circuit Design and Verification

Cathleen Heidrich

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

cathleenwork@unr.edu

Abstract

The design and verification of modern integrated circuits increasingly depend on computational methods that can reason over large, heterogeneous graphs representing netlists, timing paths, physical layout, and verification constraints. Graph neural networks have emerged as a promising class of models for learning such representations, yet their practical deployment in electronic design automation remains constrained by hardware efficiency, data movement, model robustness, and verification-grade reliability. This paper examines hardware-aware graph neural network optimization as a system-level problem rather than a narrow model-compression task. It argues that scalable circuit design and verification require coordinated choices across graph representation, accelerator architecture, memory hierarchy, partitioning strategy, distributed training infrastructure, and governance mechanisms. The discussion addresses structural trade-offs between accuracy, latency, throughput, energy consumption, and interpretability in design workflows where correctness and reproducibility are non-negotiable. It further considers deployment in cloud and edge environments, the sustainability implications of large-scale training and inference, and the fairness and policy questions that arise when proprietary electronic design automation infrastructures shape access to advanced semiconductor capabilities. Through cross-domain comparisons with machine learning systems, datacenter computing, time-sensitive networking, and safety engineering, the paper develops a forward-looking perspective on co-design, verification, and institutional accountability. The central claim is that hardware-aware graph learning can become a durable foundation for scalable circuit design only if optimization is understood as a socio-technical practice encompassing architecture, infrastructure, robustness, and governance.

Keywords

graph neural networks; integrated circuit design; electronic design automation; hardware-aware optimization; verification; scalable computing; socio-technical systems.

1. Introduction

The semiconductor industry faces a persistent structural tension that has become more acute as process technology approaches physical limits. Each new generation of integrated circuits offers greater transistor density, but the design and verification burden grows faster than the available productivity gains from conventional electronic design automation. Modern systems-on-chip contain billions of transistors, heterogeneous intellectual property blocks, multiple clock domains, complex power delivery networks, and stringent reliability requirements. Design teams must evaluate placement, routing, timing, power, thermal behavior, manufacturability, and security concurrently, often across organizational boundaries.

In this environment, graph neural networks have attracted attention because circuits are naturally represented as graphs whose nodes and edges encode devices, nets, pins, timing arcs, and physical adjacency [1], [4]. Graph convolutional and attention-based architectures can learn relational patterns that are difficult to encode with handcrafted heuristics, and inductive methods can generalize to unseen circuit instances [2], [3]. Surveys of the field have documented rapid growth in graph learning for electronic design automation, including prediction, optimization, and verification tasks [5], [8].

Yet the movement from research prototypes to production-grade design and verification has been slower than early results might suggest. The reason is not merely model accuracy. Circuit graphs are large, sparse, heterogeneous, and often irregular. Training and inference require frequent irregular memory access, substantial communication across partitions, and careful handling of dynamic graph structure. A model that performs well on a benchmark may be unusable when integrated into a commercial design flow that requires deterministic runtime, bounded memory, auditability, and seamless interaction with legacy tools. Hardware-aware optimization therefore becomes essential. It is not enough to reduce parameter counts or apply generic quantization. The optimization must account for the target accelerator, memory bandwidth, interconnect topology, batch construction, graph sampling, and the verification semantics of the design stage in which the model is deployed. Recent work on accelerating heterogeneous circuit graph training on graphics processing units illustrates both the promise and the complexity of this agenda [6].

This paper adopts a system-level perspective. It treats hardware-aware graph neural network optimization as an interdisciplinary problem spanning computer architecture, machine learning systems, electronic design automation, distributed infrastructure, and governance. The argument is that scalable circuit design and verification will not be achieved by isolated algorithmic advances. Instead, it requires co-design across graph representation, hardware mapping, scheduling, verification, and institutional policy. The sections that follow examine the foundations of hardware-aware graph learning for circuits, architectural trade-offs in accelerators and distributed systems, the socio-technical nature of design and verification workflows, deployment and sustainability concerns, robustness and fairness, and forward-looking pathways for research and practice.

2. Foundations of Hardware-Aware Graph Learning for Integrated Circuits

Graph neural networks provide a general framework for learning on relational data. Early formulations established that message passing over graph structures can approximate complex functions while respecting topological dependencies [4]. Subsequent work introduced convolutional and attention-based mechanisms that aggregate information from local neighborhoods and can be adapted to inductive settings [1], [2]. Sampling-based methods further enabled training on large graphs by approximating neighborhood aggregation without materializing full computation graphs [3]. These developments are directly relevant to integrated circuit design because circuits are graphs at multiple levels of abstraction. At the transistor level, nodes may represent devices and edges may represent electrical connectivity. At the gate level, nodes may represent logic gates and edges may represent nets. At the physical level, nodes may represent standard cells or macros, and edges may encode placement proximity, routing congestion, or thermal coupling. Verification adds another layer in which nodes represent assertions, coverage points, state elements, or traces, and edges represent temporal or logical dependencies.

Hardware awareness begins with representation. The choice of graph schema determines what can be learned, but it also determines memory footprint and parallelism. Homogeneous graphs simplify hardware mapping but lose semantic distinctions among cell types, clock domains, and power domains. Heterogeneous graphs preserve richer information but require specialized kernels and more complex memory layouts. Circuit graphs are also highly skewed: a small number of high-fanout nets may dominate communication, while many nodes have low degree. This skew creates load imbalance on parallel hardware. If the graph is partitioned naively, some processing elements may idle while others are saturated. Hardware-aware optimization must therefore consider graph partitioning, node reordering, edge grouping, and sparse format selection. Compiler frameworks for deep learning have shown that automated scheduling and kernel generation can significantly improve performance across diverse hardware targets [9]. Similar techniques are needed for graph neural networks, but circuit graphs introduce irregularity and domain-specific constraints that general-purpose compilers do not fully address.

The optimization space extends beyond individual kernels. Training a circuit graph neural network involves data ingestion from design databases, graph construction, feature extraction, sampling, aggregation, loss computation, and parameter updates. Each stage has different bottlenecks. Graph construction may be limited by input and output and database access; sampling may be limited by random memory access; aggregation may be limited by communication; and parameter updates may be limited by synchronization. Hardware-aware design must therefore optimize the pipeline rather than isolated operations. This is consistent with broader lessons from machine learning systems, where hidden technical debt accumulates when data dependencies, configuration, and monitoring are treated as afterthoughts [14]. In circuit design, such debt is especially dangerous because a model that silently drifts or consumes excessive resources can delay tape-out or introduce verification gaps.

3. Architecture and Optimization Trade-Offs in Graph Neural Accelerators

The architectural landscape for graph neural network acceleration includes general-purpose graphics processing units, specialized tensor accelerators, field-programmable gate arrays, and emerging in-memory computing substrates. Each offers different trade-offs. Graphics processing units provide high throughput and mature software ecosystems, but their performance on irregular graph workloads depends heavily on memory coalescing, occupancy, and synchronization. Specialized accelerators can exploit sparsity and mixed precision, but they may lack flexibility for rapidly evolving graph models. Field-programmable gate arrays offer reconfigurability and deterministic latency, which are attractive for verification, yet they require significant engineering effort and may struggle with very large graphs. In-memory computing promises energy efficiency for certain aggregation patterns, but manufacturing maturity and design tool support remain limited. Hardware-aware graph neural network optimization must therefore be target-specific rather than universal.

A central trade-off is between model capacity and hardware efficiency. Larger models can capture more complex circuit behavior, but they increase memory traffic and latency. In design verification, latency is not merely a user-experience concern; it affects iteration speed and the number of design alternatives that can be explored. Throughput matters for batch verification, where thousands of assertions or test cases must be evaluated. Energy consumption matters for datacenter cost and sustainability. Recent acceleration work on heterogeneous circuit graphs demonstrates that careful mapping of graph operations to

graphics processing units can reduce training time, but it also shows that gains depend on graph partitioning, kernel fusion, and communication scheduling [6]. General deep learning compilers provide useful primitives for such optimization, including operator fusion, memory planning, and auto-tuning [9], while graph substitution systems show that equivalent computations can be rewritten for better performance [10]. For circuit graphs, these techniques must be extended with domain-aware constraints, such as preserving timing semantics and avoiding transformations that obscure verification evidence.

Distributed training introduces another layer of trade-offs. Circuit graphs can be partitioned across devices by node, by edge, or by subgraph. Node partitioning may balance memory but require extensive communication for high-fanout nets. Edge partitioning may reduce communication for certain aggregation patterns but complicate feature synchronization. Parameter servers offer a flexible abstraction for distributed machine learning, but they can become bottlenecks under high contention [19]. All-reduce and collective communication strategies may perform better for dense models but less well for irregular graphs. Hardware-aware optimization must therefore consider network topology, interconnect bandwidth, and failure modes. The tail at scale problem is instructive: even a small fraction of slow nodes can dominate latency in large distributed systems [11]. In circuit design, where verification deadlines are often rigid, tail latency can translate into missed market windows or incomplete coverage.

Hardware-aware neural architecture search and deployment specialization offer one pathway for managing this complexity. Techniques such as once-for-all training allow a single supernet to be specialized for different hardware targets without retraining from scratch [20]. Applied to circuit graph learning, such methods could generate models tailored to different design stages, such as placement prediction on a datacenter accelerator and local timing estimation on an edge device. However, specialization must be governed carefully. A model optimized for one technology node or one intellectual property block may not generalize to another. Verification requires documented assumptions, validated operating ranges, and traceable model versions. Without such governance, hardware-aware optimization can produce brittle artifacts that are efficient in benchmarks but unreliable in production.

4. Design and Verification Workflows as Socio-Technical Infrastructure

Integrated circuit design and verification are not purely technical activities. They are socio-technical infrastructures involving multiple organizations, proprietary tools, open-source components, standards bodies, foundries, intellectual property vendors, and regulatory regimes. A graph neural network that predicts congestion or timing may be embedded in a flow that includes simulation, formal verification, emulation, and physical verification. Its outputs may influence decisions made by engineers who are accountable for correctness. Therefore, the optimization objective cannot be reduced to speed or accuracy. It must include explainability, reproducibility, auditability, and compatibility with existing sign-off criteria.

The infrastructure dimension is visible in the way design data moves across boundaries. A single system-on-chip may integrate blocks from different teams, each with its own naming conventions, abstraction levels, and verification environments. Graph construction must reconcile these differences without losing semantic meaning. Model training may occur in a cloud environment, while inference may occur on secure premises because of intellectual property concerns. Data governance must address access control, provenance, retention, and export restrictions. Frameworks for model cards and datasheets for datasets provide useful templates for documenting intended use, limitations, and ethical considerations [15], [16]. In

the circuit domain, such documentation could include technology node, temperature range, supply voltage assumptions, verification coverage, and known failure modes. This is not bureaucratic overhead; it is a precondition for trust in automated design assistance.

Verification workflows add further constraints. Formal verification seeks proof-like guarantees, while simulation and emulation provide statistical evidence. A graph neural network may be used to prioritize verification efforts, predict coverage gaps, or guide test generation. If the model is wrong, the consequences may include escaped bugs, costly respins, or safety failures. Robustness and uncertainty quantification are therefore essential. The model should signal when it is operating outside its training distribution and should provide evidence that can be checked by conventional tools. This aligns with broader concerns in artificial intelligence safety, where specification, robustness, and oversight are central challenges [18]. Hardware-aware optimization must preserve or enhance these properties. For example, quantization may reduce memory usage but also reduce the fidelity of uncertainty estimates. Pruning may improve latency but remove neurons that are critical for rare corner cases. The optimization process must therefore be evaluated against verification metrics, not only standard machine learning benchmarks.

5. Deployment, Scalability, and Sustainability

Deployment choices shape the scalability and sustainability of graph neural network optimization for circuit design. Centralized cloud deployment offers elastic compute, shared storage, and access to specialized accelerators. It simplifies large-scale training and allows teams to collaborate across geographies. However, it raises concerns about data sovereignty, network bandwidth, and dependency on external providers. Datacenter-scale computing has its own architectural logic, including warehouse-scale machine design, fault tolerance, and energy management [12]. Graph neural network workloads must fit into this logic. They may require high-bandwidth interconnects, large memory pools, and scheduling policies that account for irregular communication. Time-sensitive networking standards and deterministic networking research provide lessons for bounding latency and jitter in distributed design flows [17]. If verification jobs must complete within a fixed window, best-effort networking may be insufficient.

Edge deployment offers an alternative for latency-sensitive or confidentiality-sensitive tasks. Edge computing brings computation closer to data sources, reducing wide-area communication and enabling local control [13]. In circuit design, edge deployment might support in-fab metrology, real-time failure analysis, or secure verification on premises. Yet edge devices have limited memory, power, and thermal budgets. Hardware-aware optimization must therefore compress models, schedule operations efficiently, and manage intermittent connectivity. Hybrid architectures may be most practical: heavy training and global optimization occur in the cloud, while inference and adaptation occur at the edge. Such hybrids require careful synchronization and governance. They also raise questions about who controls the model and who can audit its behavior.

Sustainability is an increasingly important dimension. Large-scale training of graph neural networks consumes energy and contributes to carbon emissions, both directly through computation and indirectly through cooling and infrastructure. Inference at scale can also be significant, especially if models are retrained frequently or deployed across many design projects. Hardware-aware optimization can reduce energy per inference, but it may also encourage more computation through rebound effects. If faster models enable more design iterations, total energy use may rise even as per-operation efficiency improves. A sustainable

strategy must therefore consider lifecycle assessment, workload scheduling, and the use of renewable energy. It must also consider the opportunity cost of computation: energy spent on exploring marginal design alternatives may be better allocated to verification or to designs with greater societal benefit. These are not purely technical decisions; they involve organizational priorities and policy incentives.

6. Robustness, Fairness, Governance, and Policy

Robustness is a first-order requirement for graph neural networks in circuit design and verification. Circuit graphs are subject to distribution shift when moving from one technology node to another, from one foundry to another, or from one design style to another. Adversarial or accidental perturbations in features can alter predictions. Sparse and skewed graph structures can cause unstable aggregation. A robust system must include outlier detection, uncertainty estimation, fallback mechanisms, and continuous monitoring. It must also be resilient to hardware faults. If a specialized accelerator produces silent data corruption, the resulting design decisions may be invalid. Fault tolerance and error detection are therefore part of hardware-aware optimization, not separate concerns.

Fairness enters the discussion through access and allocation. Advanced electronic design automation tools, proprietary process design kits, and large-scale compute resources are not equally available to all organizations, regions, or researchers. If graph neural network optimization depends on privileged access to data and hardware, it may widen existing gaps in semiconductor capability. Smaller firms, academic groups, and emerging economies may struggle to compete. Open benchmarks, reproducible models, and shared infrastructure can mitigate this imbalance, but they require governance. Model cards and datasheets can improve transparency, but they do not by themselves ensure equitable access [15], [16]. Policy frameworks for export control, intellectual property protection, and national security also shape what can be shared. There is a tension between openness as a driver of innovation and restriction as a safeguard for security. A balanced approach would distinguish between generic research artifacts and sensitive design data, while investing in verification and assurance methods that make sharing safer.

Governance must also address accountability. When a graph neural network recommends a placement or flags a verification gap, who is responsible if the recommendation is wrong? The engineer, the tool vendor, the model developer, or the organization that deployed it? Clear lines of responsibility are needed, along with audit trails and version control. Verification evidence should be reproducible, and model updates should be traceable. Standards bodies could play a role by defining interfaces, documentation requirements, and benchmark suites. The broader artificial intelligence safety literature emphasizes the importance of oversight, interpretability, and robust specification [18]. Circuit design provides a domain where these principles can be operationalized because the cost of failure is high and the verification culture is already mature. Hardware-aware optimization should therefore be designed with governance hooks, such as logging, explainability, and human-in-the-loop review, from the beginning.

7. Case Illustrations and Cross-Domain Comparisons

Several case illustrations clarify the system-level argument. In physical design, graph placement methodologies have demonstrated that learned models can propose chip layouts that meet or exceed human-designed solutions on certain metrics [7]. These results are significant, but they also show the importance of integration with commercial place-and-route

tools, timing analysis, and design rule checking. The model does not replace the flow; it augments it. Similarly, machine learning for electronic design automation spans prediction, optimization, and verification tasks, each with different data requirements and failure modes [8]. A hardware-aware graph neural network for placement may prioritize throughput and memory capacity, while a model for formal verification may prioritize determinism and evidence generation. The optimization objective must be aligned with the downstream decision.

Cross-domain comparisons are instructive. Graph neural networks were first popularized in social network analysis and citation networks, where large graphs are also common [1], [2]. However, social graphs are noisy and probabilistic, while circuit graphs are engineered and governed by strict semantics. A wrong prediction in a social recommendation may be inconvenient; a wrong prediction in timing verification may cause a chip to fail. This difference raises the bar for robustness and interpretability. Edge computing provides another comparison. In Internet of Things deployments, edge intelligence must balance latency, privacy, and energy [13]. In circuit manufacturing, edge intelligence could support real-time process control or secure defect classification. The constraints are similar, but the stakes are higher because manufacturing deviations can affect millions of devices.

Time-sensitive networking offers a third comparison. Industrial control systems require bounded latency and reliable delivery, which has driven standards for deterministic networking [17]. Circuit design flows increasingly rely on distributed compute and remote collaboration, so they face analogous requirements. If a verification job is delayed by network congestion, the entire project schedule may slip. Hardware-aware optimization should therefore consider end-to-end latency, not just accelerator time. Distributed machine learning frameworks have shown that parameter synchronization and communication can dominate performance [19]. In circuit design, those effects are amplified by data locality and security constraints. A model may need to be trained across multiple sites without moving sensitive intellectual property. Federated or privacy-preserving techniques may help, but they introduce additional communication and governance complexity.

8. Forward-Looking Perspectives

The next stage of research should move toward co-design across layers. Graph representation, model architecture, compiler scheduling, accelerator design, and verification should be optimized together rather than sequentially. This requires shared abstractions and benchmarks that capture circuit-specific semantics. It also requires hardware-aware neural architecture search and deployment specialization to be integrated with design flow constraints. Techniques such as once-for-all specialization show that a single model family can serve multiple hardware targets [20]. Compiler frameworks such as TVM and TASO provide foundations for automated optimization and graph rewriting [9], [10]. Extending these ideas to heterogeneous circuit graphs will require new intermediate representations, domain-specific cost models, and verification-aware transformations.

Another direction is the development of verification-grade machine learning. Models should produce not only predictions but also certificates, confidence bounds, and explanations that can be checked by conventional tools. This may involve hybrid approaches that combine graph neural networks with symbolic reasoning, formal methods, and simulation. It may also involve runtime monitors that detect distribution shift or hardware faults. The safety literature offers principles for specification, robustness, and oversight [18], but circuit design offers a concrete setting where these principles can be tested under industrial constraints.

Documentation practices such as model cards and datasheets should be adapted to include verification coverage, technology assumptions, and known limitations [15], [16].

Infrastructure and policy must evolve alongside technical advances. Cloud and edge architectures will continue to blur, requiring new standards for data governance, latency, and security. Time-sensitive networking and deterministic communication can provide building blocks for distributed design flows [17]. Sustainability must be measured and managed, not assumed. Fairness must be considered in access to models, data, and compute. Governance must assign responsibility and ensure auditability. In this sense, hardware-aware graph neural network optimization is not only a technical program but also an institutional one. It asks how societies can design, verify, and deploy increasingly complex computational artifacts in ways that are efficient, reliable, equitable, and accountable.

9. Conclusion

Hardware-aware graph neural network optimization for scalable integrated circuit design and verification is a multidimensional challenge. Graph neural networks offer powerful representational capabilities for circuits, but their value depends on how well they are mapped to hardware, integrated into design flows, and governed across organizations. This paper has argued that optimization must be understood as a system-level practice encompassing graph representation, accelerator architecture, distributed infrastructure, deployment models, sustainability, robustness, fairness, and policy. The required acceleration techniques for heterogeneous circuit graphs demonstrate that significant performance gains are possible, but they also reveal the depth of co-design required. The path forward lies in shared benchmarks, verification-aware model design, transparent documentation, equitable access, and accountable governance. If these elements are pursued together, hardware-aware graph learning can become a durable foundation for the next generation of integrated circuit design and verification.

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. Luo, Y., Li, S., Tao, J., Thorat, K. G., Xie, X., Peng, H., ... & Huang, S. (2025, June). Dr-circuitgnn: Training acceleration of heterogeneous circuit graph neural network on gpus. In Proceedings of the 39th ACM International Conference on Supercomputing (pp. 221-235).

7. Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., ... & Dean, J. (2021). A graph placement methodology for fast chip design. *Nature*, 594(7862), 207-212.
8. Huang, G., Hu, J., He, Y., Liu, J., Ma, M., Shen, Z., ... & Zhang, Z. (2021). Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 26(5), 1-46.
9. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., ... & Ceze, L. (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).
10. Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., & Aiken, A. (2019). TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (pp. 47-62).
11. Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, 56(2), 74-80.
12. Barroso, L. A., & Hölzle, U. (2009). The datacenter as a computer: An introduction to the design of warehouse-scale machines. Morgan & Claypool.
13. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637-646.
14. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems* (pp. 2503-2511).
15. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., ... & Gebru, T. (2019). Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (pp. 220-229).
16. Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, 64(12), 86-92.
17. Nasrallah, A., Thyagaturu, A. S., Alharbi, Z., Wang, C., Shao, X., Reisslein, M., & ElBakoury, H. (2019). Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Communications Surveys & Tutorials*, 21(1), 88-145.
18. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.
19. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., ... & Su, B. Y. (2014). Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation* (pp. 583-598).
20. Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.