Poster Title

Quantum Simulators and Applications on Quantum Framework

Poster Authors

Srikar Chundury	schundu3@ncsu.edu
Zhihao Xu	zxu8@nd.edu
Amir Shehata	shehataa@ornl.gov
Seongmin Kim	kims@ornl.gov
Frank Mueller	fmuelle@ncsu.edu
In-Saeng Suh	suhi@ornl.gov

Poster Abstract

Simulating quantum circuits is essential for validating quantum algorithms. However, no single simulator consistently performs best — efficiency depends on circuit structure, entanglement, and depth. In this work, we integrate Qiskit-Aer (state-vector and matrix product state) and QTensor, a tree-tensor-network based simulator, into the Quantum Framework (QFw), a modular platform that supports multiple quantum backends via a unified interface. We also enable distributed quantum approximate optimization algorithm (DQAOA) application compatibility with QFw, allowing sub-problems to be solved in parallel at scale. We then benchmark DQAOA and TFIM (transverse field Ising model) circuits across supported simulators, showing how performance varies significantly with problem type. All simulations are deployed on the Frontier supercomputer using QFw's MPI-based orchestration for distributed, multi-node execution. These results underscore the need for simulator-agnostic infrastructure to enable systematic evaluation and high-performance scaling of quantum workloads. QFw provides a practical and extensible path toward reproducible quantum algorithm development across diverse application domains.

Poster Relevance

- Quantum Software Engineering
- Quantum Computing

Quantum Simulators and Applications on Quantum Framework

Srikar Chundury*†, Zhihao Xu*‡, Amir Shehata*, Seongmin Kim*, Frank Mueller†, In-Saeng Suh*
*Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA
†North Carolina State University, Raleigh, NC 27695, USA
‡University of Notre Dame, Notre Dame, IN 46556, USA

***Abstract*—Simulating quantum circuits is essential for validating quantum algorithms. However, no single simulator consistently performs best — efficiency depends on circuit structure, entanglement, and depth. In this work, we integrate Qiskit-Aer (state-vector and matrix product state) and QTensor, a tree-tensor-network based simulator, into the Quantum Framework (QFw), a modular platform that supports multiple quantum backends via a unified interface. We also enable distributed quantum approximate optimization algorithm (DQAOA) application compatibility with QFw, allowing sub-problems to be solved in parallel at scale. We then benchmark DQAOA and TFIM (transverse field Ising model) circuits across supported simulators, showing how performance varies significantly with problem type. All simulations are deployed on the Frontier supercomputer using QFw's MPI-based orchestration for distributed, multi-node execution. These results underscore the need for simulator-agnostic infrastructure to enable systematic evaluation and high-performance scaling of quantum workloads. QFw provides a practical and extensible path toward reproducible quantum algorithm development across diverse application domains.**

***Index Terms*—Quantum Computing, Quantum Software Engineering, Tensor Networks, Quantum Circuit Simulation**

A. Introduction & Background: Quantum circuit simulation plays a central role in developing and validating quantum algorithms on HPC systems. While many simulators exist, their performance varies significantly depending on circuit structure, entanglement, and depth, making it challenging to select the optimal backend for a given workload. A unified benchmarking approach is thus crucial to guide efficient simulator selection and to advance scalable quantum algorithm development. The Quantum Framework (QFw) [1], developed at Oak Ridge National Laboratory, is a modular HPC quantum framework designed to integrate quantum simulators and hardware under a unified API. QFw decouples simulator implementations from application logic, supporting backend-agnostic quantum application development with seamless switching between simulators. Built atop the PRTE Distributed Virtual Machine (PRTE DVM) [1], [2], QFw enables scalable multi-node deployment with fault tolerance and rapid job spawning. QFw's flow involves application frontend invocation, where users write quantum applications using frontends such as Qiskit [3] or Pennylane to construct quantum circuits, optimization problems, or time evolution workflows for HPC execution. The QFWBackend [2] is a lightweight Python interface, translating standard circuit definitions and execution calls into QFw API requests to the Quantum Platform Manager (QPM) that then selects the appropriate simulator backend, allocates computational resources across available nodes, and manages simulator instantiation, decoupling simulator choice from application logic for seamless deployment. The Quantum Resource Controller (QRC) orchestrates task scheduling across nodes and processes, ensuring efficient use of HPC resources by coordinating execution order, monitoring job status, and managing dependencies. The Distributed Execution Framework (DEFw) facilitates Remote Procedure Calls (RPCs) between QFw services, such as QPM and QRC and the backend simulators, abstracting inter-component communication to enable scalable, low-latency interactions across the distributed system via PRTE-DVM. Backend simulators then execute these quantum circuits. Finally, QFw aggregates simulation results and returns them to the application frontend, allowing users to retrieve expectation values, measurement samples, or final state vectors without managing backend-specific data handling. In this way, QFw enables systematic benchmarking of quantum applications, aiding efforts in quantum advantage candidate search.

This work extends QFw by integrating it with Qiskit-Aer [3] and QTensor [4], and by demonstrating its capabilities for benchmarking quantum workloads such as TFIM and DQAOA [5] [6] at scale on HPC systems.

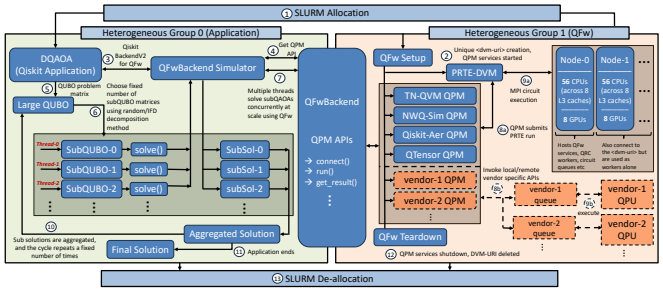


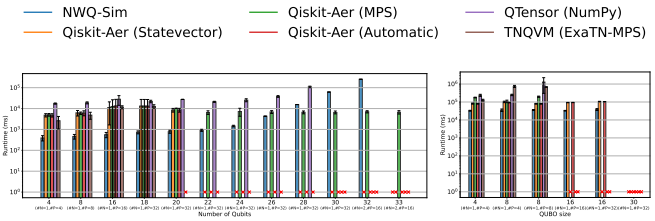
Fig. 1: DQAOA + QFw + SLURM Workflow.

B. Approach: After integrating Qiskit-Aer [3] and QTensor [4] into QFw, it can now support a variety of simulations such as state-vector, matrix product state (MPS) and tree tensor-network (TTN) methods. Integration involved implementing their respective QPM API interfaces (step-3, step-

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.(http://energy.gov/downloads/doe-public-279 access-plan).

4), leveraging QFw's extensible design to manage backend-specific configuration, execution, and result retrieval while remaining transparent to frontend application code. To support scalable quantum optimization, we enabled DQAOA application compatibility with QFw. Our workflow (Fig. 1) begins with a dual-group heterogeneous SLURM allocation (step-1), one group for DQAOA and another for QFw. In hetgroup-1, QFw creates a unique PRTE-DVM URI and starts QPM services (step-2). In hetgroup-0, the Qiskit-Python-based DQAOA application starts, taking the quadratic unconstrained binary optimization (QUBO) matrix as input (step-5). While this happens, the QFwBackend establishes secure connections to the QPM services (step-4). The large QUBO is then decomposed into multiple subQUBOs using random or impact factor directed (IFD) decomposition methods (step-6), and Python's threading package issues concurrent subQUBO `solve` calls. We opted for threading because our workload is I/O bound, as each call makes an asynchronous RPC to QFw (step-7) to trigger circuit execution. These RPC calls involve network communication to QPM services, which submit jobs to PRTE (step-8) and launch MPI-based circuit executions (step-9) across allocated HPC cores and nodes. Sub-solution results are collected and aggregated (step-10) before the optimization cycle repeats. After the application converges, the QPM services are shut down, the DVM URI is deleted and SLURM resources are deallocated (step-13). This design enables asynchronous backend invocation using HPC parallelism, solving DQAOA efficiently at scale.

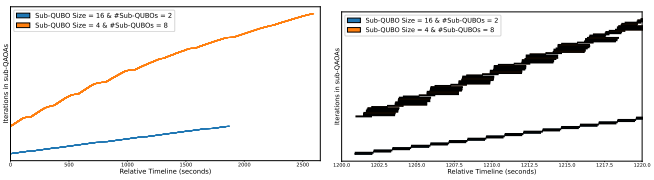
C. Tests and Results: We benchmarked multiple simulators (Qiskit-Aer, NWQ-Sim [7], TN-QVM [8], and QTensor) for TFIM, QAOA, and DQAOA circuits. Figure 2a shows TFIM scaling simulation performance, where Qiskit-Aer's matrix product state (MPS) method efficiently solves up to 33-qubit TFIM circuits, while other simulators do not scale to such problem sizes. We observe that NWQ-Sim also solves TFIM-32, although with higher runtime compared to Qiskit-Aer's MPS. For QAOA benchmarks, Figure 2b presents scaled optimization runs. Some data is missing because simulators took longer than two hours to solve larger QUBO problems, when execution was cut off.



(a) (b)

Fig. 2: Simulation performance results: (a) TFIM, and (b) QAOA. Above, #N is number of nodes, and #P is number of processes per node.

As seen in Figures 3a and 3b, DQAOA can benefit from scaling each circuit execution within subQAOAs using QFw. This provides developers with more detail about the behavior of such an algorithm at scale. We also noted that with smaller



(a) (b)

Fig. 3: DQAOA timing analysis with NWQ-Sim backend: (a) full timing results for various subQUBO configurations, and (b) zoomed view showing concurrent subQAOAs.

subQUBOs, each subQAOA took more iterations for completion compared to larger subQUBOs. These results demonstrate that simulator performance varies significantly based on circuit type, entanglement, and problem size, underscoring the importance of simulator-agnostic frameworks such as QFw for scalable and reproducible quantum algorithm development on HPC systems.

E. Future Work: QFw could benefit from automatic QPM selection based on circuit metadata to select optimal simulators and execute them at scale. Also, we intend to explore hybrid HPC/QC execution of DQAOA, where subQUBOs are dynamically mapped across simulators and actual hardware (as shown via dotted lines in Figure 1) for performance portable quantum workflows.

Acknowledgment: This work was supported in part by NSF awards PHY-1818914, MPS-2120757, CCF-2217020, CISE-2316201, and PHY-2325080 as well as DOE DE-SC0025384. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] A. Shehata, T. Naughton, and I.-S. Suh, "A framework for integrating quantum simulation and high performance computing," 2024. [Online]. Available: https://arxiv.org/abs/2408.08098

[2] S. Chundury, A. Shehata, T. Naughton III, S. Kim, F. Mueller, and I.-S. Suh, "QFw: A quantum framework for large-scale hpc ecosystems." Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), 11 2024. [Online]. Available: https://www.osti.gov/biblio/2498439

[3] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.08810

[4] D. Lykov, A. Chen, H. Chen, K. Keipert, Z. Zhang, T. Gibbs, and Y. Alexeev, "Performance evaluation and acceleration of the qtensor quantum circuit simulator on gpus," 2021. [Online]. Available: http://dx.doi.org/10.1109/QCS54837.2021.00007

[5] Z. Xu, S. Chundury, S. Kim, A. Shehata, X. Li, A. Li, T. Luo, F. Mueller, and I.-S. Suh, "GPU-Accelerated Distributed QAOA on Large-scale HPC Ecosystems," 6 2025. [Online]. Available: https://arxiv.org/abs/2506.10531

[6] S. Kim, T. Luo, E. Lee, and I.-S. Suh, "Distributed quantum approximate optimization algorithm on integrated high-performance computing and quantum computing systems for large-scale optimization," 2024. [Online]. Available: https://arxiv.org/abs/2407.20212

[7] A. Li, B. Fang, C. E. Granade, G. Prawiroatmodjo, B. Heim, M. Rötteler, and S. Krishnamoorthy, "rm SV-Sim: Scalable pgas-based state vector simulation of quantum circuits," 2021.

[8] A. J. McCaskey, "Tensor Network Quantum Virtual Machine (TNQVM)," 11 2016. [Online]. Available: https://www.osti.gov//servlets/purl/1340180

Quantum Simulators and Applications on Quantum Framework

Srikar Chundury[1,2], Zhihao Xu[1,3], Amir Shehata[1], Seongmin Kim[1], Frank Mueller[2], and In-Saeng Suh[1]

[1] Oak Ridge National Laboratory, Oak Ridge, TN 37830, [2] North Carolina State University, Raleigh, NC 27606, [3] University of Notre Dame, Notre Dame, IN 46556

Motivation

- Quantum circuit simulation is essential for validating quantum algorithms
- No single simulator is universally optimal
  - Performance depends on problem structure, entanglement, circuit depth, number of connected components, and maybe more (?)
- Systematic benchmarking is required to identify the most efficient simulator for each workload/circuit
- Therefore, scalable, simulator-agnostic frameworks are needed to identify quantum advantage candidates

Background

Quantum Framework (QFw) [1–3]

- QFw is a modular HPC quantum framework developed at ORNL to integrate quantum simulators and hardware in a unified API → see component workflow in Fig. 1
- Supports backend-agnostic quantum application development
- Built atop PRTE-DVM to allow scalable multinode deployment with fault tolerance and rapid job spawning

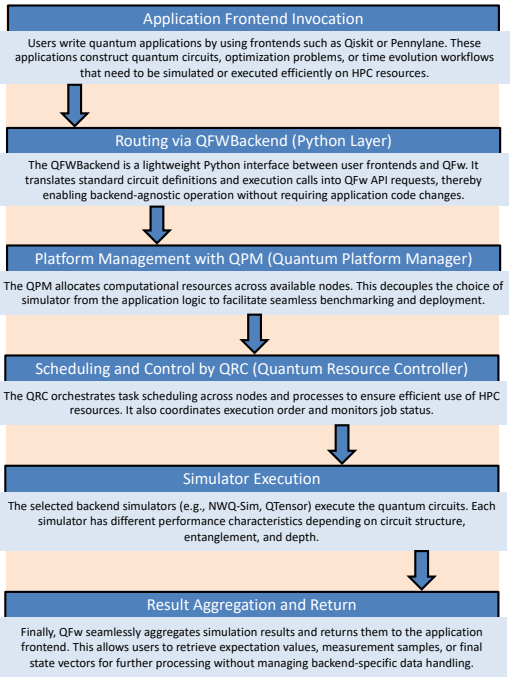


Fig. 1. QFw components.

- QFw uses the Distributed Execution Framework (DEFw) to enable scalable, low-latency RPC communication between QPM, QRC, and backend simulators
  - DEFw → Infrastructure that abstracts inter-component communication via PRTE-DVM without being part of the application or QPM layers directly

Quantum Algorithms (Relevant for this Work)

- TFIM (Transverse-Field Ising Model)
  - Models quantum spin systems with both interaction (Ising) and transverse field terms
  - Used for benchmarking simulator performance across varying problem sizes and connectivity types
- QAOA (Quantum Approximate Optimization Algorithm) [4]
  - A gate-based variational quantum algorithm designed to solve combinatorial optimization problems such as QUBO
  - Used here for metamaterial design optimization by leveraging hybrid HPC-QC workflows
- DQAOA (Distributed QAOA) [1, 9]
  - Extends QAOA by partitioning large problems into subproblems (subQUBOs) that can be solved in parallel across HPC nodes, thereby enabling scalability beyond single simulator memory limits

This Work

- Integrated Qiskit-Aer [7] QPM into QFw
  - Methods → state-vector, matrix-product-state, and automatic
- Integrated QTensor [8] QPM into QFw
  - Support for tree-tensor-network (TTN) based simulation to efficiently handle circuits with low entanglement and high depth
- DQAOA [9] application compatibility with QFw
  - Allows partitioned subQUBOs to be executed in parallel by using QFw's PRTE
- Evaluated circuit simulation performance characteristics
  - Benchmarked multiple simulators via QFw's QPMs-TN-QVM [6], NWQ-Sim [5], Qiskit-Aer, and QTensor
  - TFIM evolution (up to 33 qubits)
  - DQAOA optimization (QUBO size 30)
- Performed scaling tests on Frontier supercomputer with larger subQUBOs for DQAOA without requiring any additional application-level code modifications
  - Integrates seamlessly with SLURM job workflows → Fig. 2
  - QFw abstracts resource management and backend execution

- Ran DQAOA-30 via QFw + NWQ-Sim for different configurations of subQUBOs for 10 cycles using the IFD decomposition method
  - SLURM allocation of 2 Frontier nodes (1 for DQAOA and 1 for QFw). Each process is tied to 1 core via PRTE
- QFw enables easy scaling → Fig. 5
  - Each circuit execution automatically uses MPI
  - At any point in the execution, #subQUBOs would be running concurrently → Fig. 6

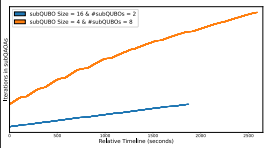


Fig. 5. DQAOA with QFw+NWQ-Sim.

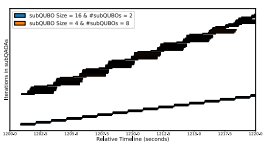


Fig. 6. Region of interest from Fig. 5.

- DQAOA solved by using NWQ-Sim demonstrates—at scale—the performance behavior of a distributed quantum algorithm
- #Iterations of smaller subQAOAs >> larger subQAOAs
- COBYLA optimizer is thread-unsafe
  - Running it via mpi4py would solve this problem but remains as future work
- Used SPSA optimizer for concurrent subQAOA execution across threads

DQAOA + QFw + SLURM

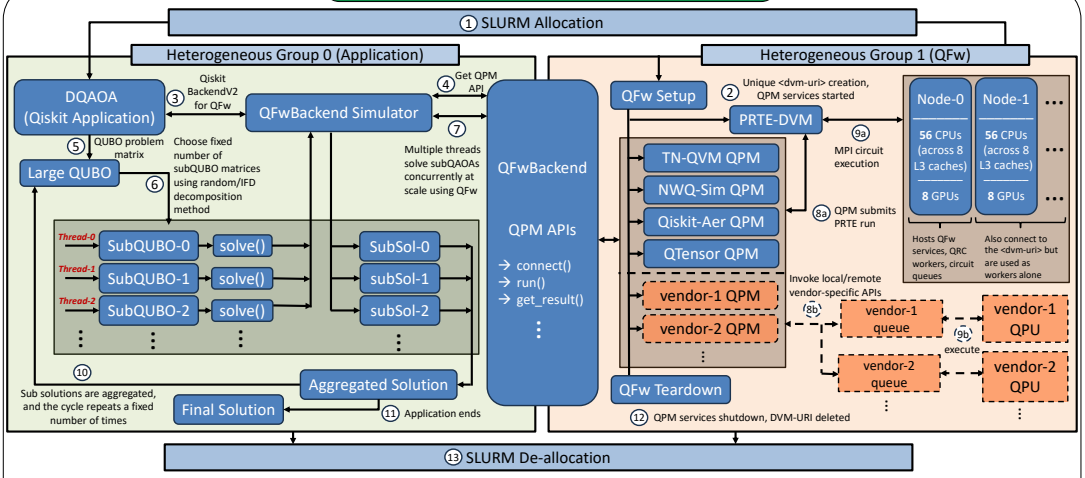


Fig. 2. Solving DQAOA using QFw.

- In Fig. 2, steps ① to ⑬ inform the workflow of how DQAOA is run using QFw
  - Green → Hetgroup 0, Orange → Hetgroup 1
  - Each parameterized ansatz from each subQAOA runs at scale using MPI via QFw (PRTE DVM-URI)

Preliminary Results

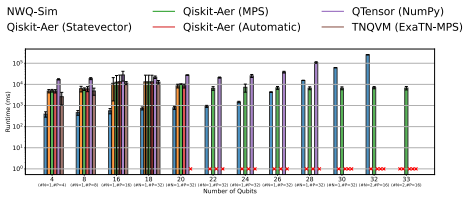


Fig. 3. TFIM scale tests.

- Single-node Qiskit-Aer's MPS can solve 33-qubit TFIM, although other simulators do not scale → Fig. 3
- Multinode NWQ-Sim was able to solve TFIM-32, although it was slower than Qiskit-Aer's MPS

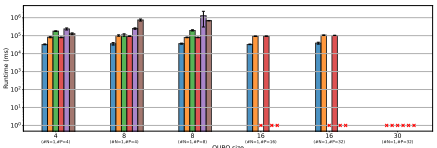


Fig. 4. QAOA with QFw.

- Tensor-Network simulators took over 2 hours to solve QAOA (QUBO >= 16) → Fig. 4

Future Work

- Hybrid HPC/QC execution of DQAOA (e.g., different subQUBOs could be solved using different QPMs)
  - → Some on actual hardware as well
- QPMs can be auto-selected for solving a particular quantum circuit depending on its metadata (width, depth, number of connected components)
  - → QFw could make that selection automatically

References

[1] Xu, Z., Chundury, S., Kim, S., Shehata, A., Li, X., Li, A., Luo, T., Mueller, F., and Suh, I.-S. (2025). GPU-Accelerated Distributed QAOA on Large-scale HPC Ecosystems. https://arxiv.org/abs/2506.10531

[2] Shehata, A., Groszkowski, P., Naughton, T. J., Gopalakrishnan Meena, M., Wong, E. Y., Chaves Claudino, D., Ferreira Da Silva, R., and Beck, T. L. (2025). Bridging paradigms: Designing for HPC-Quantum convergence. Future Generation Computer Systems, 174, 107980. https://www.sciencedirect.com/science/article/pii/S0167739X25002754

[3] Chundury, S., Shehata, A., Naughton, T. J., Kim, S., Mueller, F., and Suh, I.-S. (2024). QFw: A Quantum Framework for Large-scale HPC Ecosystems. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2024. https://www.osti.gov/biblio/2498439

[4] Kim, S., and Suh, I.-S. (2024). Performance Analysis of an Optimization Algorithm for Metamaterial Design on the Integrated High-Performance Computing and Quantum Systems. Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE 2024). https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10821180

[5] Li, A., Fang, B., Granade, C. E., Prawiroatmodjo, G., Heim, B., Rötteler, M., and Krishnamoorthy, S. (2021). SV-Sim: Scalable PGAS-based State Vector Simulation of Quantum Circuits. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021.

[6] McCaskey, A. J. Tensor Network Quantum Virtual Machine (TNQVM). Computer software. https://www.osti.gov/servlets/purl/1340180. Vers. 00. USDOE. 18 Nov. 2016. Web.

[7] Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., and Gambetta, J. M. (2024). Quantum computing with Qiskit. https://doi.org/10.48550/arXiv.2405.08810

[8] Lykov, D., Chen, A., Chen, H., Keipert, K., Zhang, Z., Gibbs, T., and Alexeev, Y. (2021). Performance Evaluation and Acceleration of the QTensor Quantum Circuit Simulator on GPUs. Proceedings of the IEEE/ACM 2nd International Workshop on Quantum Computing Software (QCS 2021). http://dx.doi.org/10.1109/QCS54837.2021.00007

[9] Kim, S., Luo, T., Lee, E., and Suh, I.-S. (2024). Distributed Quantum Approximate Optimization Algorithm on Integrated High-Performance Computing and Quantum Computing Systems for Large-Scale Optimization. https://arxiv.org/abs/2407.20212

This work was supported in part by NSF awards PHY-1818914, MPS-2120757, CCF-2217020, CISE-2316201, and PHY-2325080 as well as DOE DE-SC0025384. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725.