

Quantum Emulator Benchmarking Made Easy

**Anna Leonteva,
Maxime Outteryck,
Guido Masella**

QPerfect, 23 Rue du Loess,
67000 Strasbourg, France

Introduction

Classical simulation is a core technology for quantum computing research and development, driving progress on both hardware and software innovation. On the hardware side, emulators allow researchers to pinpoint performance bottlenecks, optimize quantum architectures, and validate error-correction protocols before physical implementation (Arute et al. 2019; Kim et al. 2023; Cross et al. 2019). In software, emulators provide an environment for the design, testing, and refinement of quantum algorithms (Morvan et al. 2024; King et al. 2025).

Since classical emulation methods are continuously evolving, they demand systematic evaluation not as a mere leaderboard competition, but as a scientific methodology that bridges quantum computing and its potential applications by revealing each emulator's strengths and limitations. Moreover, robust benchmarking can systematically map which classes of quantum circuits today's emulators can tractably simulate and which remain beyond their reach, providing a clear roadmap for future research in quantum algorithms (Leonteva et al. 2025; Jamadagni et al. 2024).

However, benchmarking quantum emulators is a challenging task, because their performance depends on many factors. A key factor is the core simulation method. For instance, statevector methods typically faces exponential demands in computational time and memory with respect to the number of simulated qubits, restricting their effective use to systems of a few tens of qubits (Jamadagni et al. 2024). Whereas, Matrix Product States (MPS) methods can achieve polynomial scaling in time and memory, allowing for simulating hundreds or even thousands of qubits for circuits with limited entanglement. Secondly, emulator-specific parameters (e.g., bond dimension and accuracy cutoffs in MPS) can strongly influence performance, depending on the circuit

(Leonteva et al. 2025). Furthermore, the programming language used to implement an emulator can significantly affect its performance. For instance, Python implementations are typically slower than those in C++, while Julia often achieves competitive run times thanks to just-in-time (JIT) compilation and efficient memory management. Finally, circuits themselves can affect results: e.g., the same emulator may scale poorly on one circuit, but achieve up to thousands of qubits on another (Leonteva et al. 2025; Jamadagni et al. 2024). All of these factors introduce biases and lead to unreliable benchmarking results (Lubinski et al. 2024). This implies that quantum emulator performance should not be characterized as a singular "empirical" property, but rather as a set of metrics (including run time, fidelity, and scalability) that are dependent on all mentioned factors.

Several open-source projects have addressed quantum emulator benchmarking from different angles. QUARK (Finžgar et al. 2022) is a modular framework automating application-level benchmarking, from problem definition to solver execution. However, its tight coupling to domain-specific encodings and limited backend support restricts its adaptability for broader, scalable comparisons. Jamadagni et al. (Jamadagni et al. 2024) presented a toolchain for statevector simulators, profiling single- vs double-precision and single- vs multi-threaded regimes across leading libraries. BenchPress (Nation et al. 2025) measures circuit construction and transpilation performance across seven SDKs, but omits simulator execution metrics. Bowles et al. (Bowles et al. 2024) introduced a PennyLane-based benchmark, reporting only machine learning quality metrics. While each tool excels in its niche — application workflows, simulator profiling, compiler performance, or quantum machine learning evaluation — none, to our knowledge, offers unified access to multiple quantum emulators with execution benchmarking across diverse circuits. Our framework addresses this gap.

The goal of this paper is to present an open-source framework, FENIQS¹, for automated benchmarking of quantum emulators across simulation methods, circuit types, and user-defined scenarios. It aims to streamline the benchmarking process, reducing time and manual effort involved.

¹ https://github.com/qperfect-io/feniqs_lite

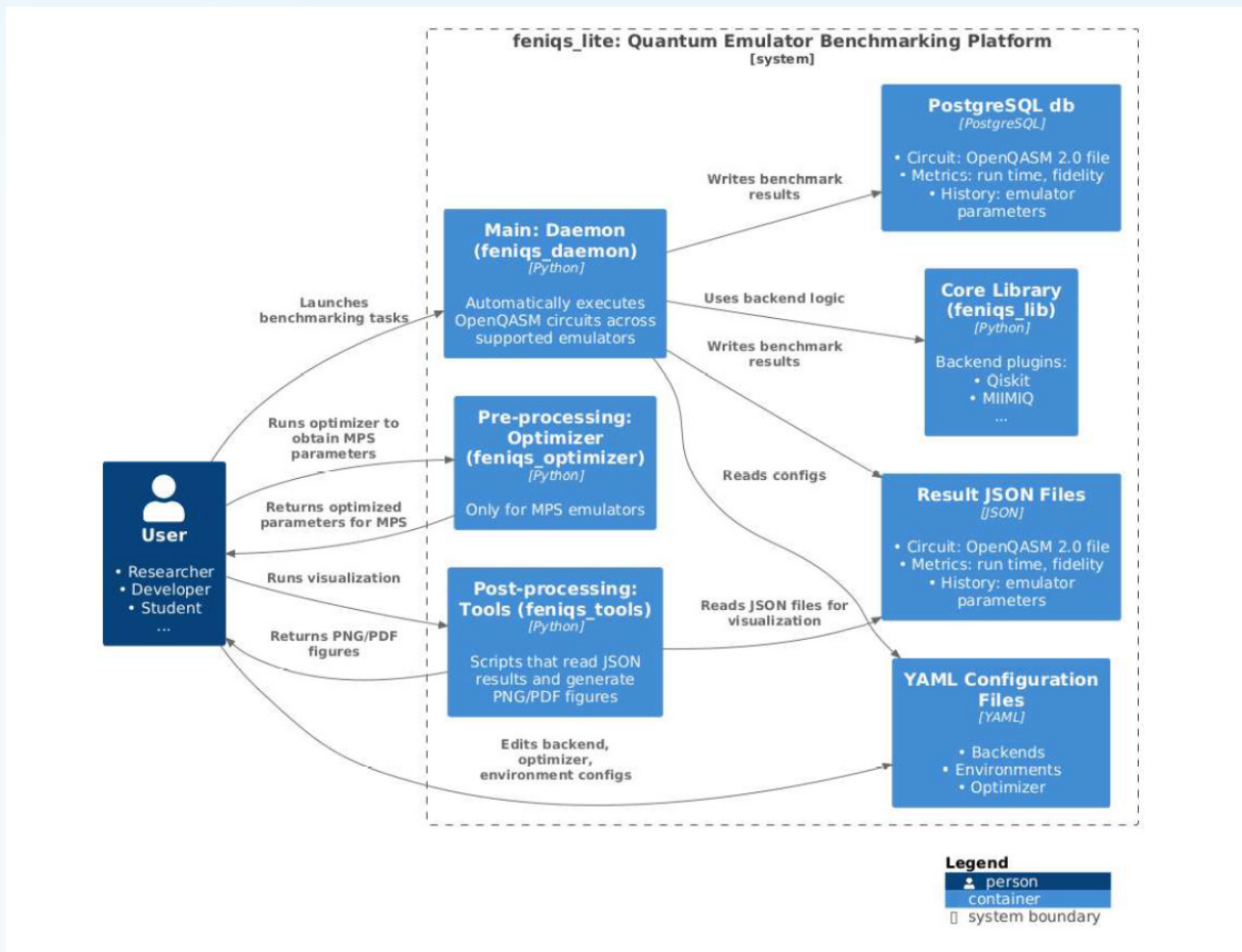


Figure 1: Architectural diagram of the FENIQS benchmarking platform, illustrating key components, data stores, and user interactions. The large dashed rectangle delineates the system boundary of FENIQS. Boxes within this boundary represent primary software containers and data stores, while boxes positioned externally indicate persons interacting with the system. Arrows indicate the direction of control signals or data flows.

Framework

FENIQS is an open-source Python framework designed to streamline the quantum emulator benchmarking process. Its architecture is user-centric, problem-agnostic, and language-independent, featuring a modular and extensible design, where all supported emulators are implemented as plugins. FENIQS benchmarking process is shown in Figure 1 and contains several principal blocks.

Workflow: Users initiate this workflow by supplying quantum circuits, in OpenQASM 2.0 format, to a designated input directory, which is monitored by a FENIQS daemon (Linux service). Each new circuit is processed by systematically invoking the configured emulator plugins via the `feniqs_lib` core library. Throughout the execution of each simulation, the framework rigorously enforces user-defined timeouts. Upon completion of a simulation run, all pertinent performance metrics, such as run time and fidelity (discussed in Section 3), along with associated metadata, are captured and stored. FENIQS provides flexibility in data archiving, offering options for storage in a PostgreSQL

database, facilitating robust and queryable long-term storage, or as individual JSON files. These JSON files can then feed into the framework's post-processing tools to generate a variety of insightful visualizations, including comparative plots of run time, fidelity, and scalability across different emulators and configurations.

Core Library: FENIQS currently supports twelve actively maintained quantum emulator frameworks, covering a wide range of simulation methods and hardware targets (summarized in Table 1). Internally, the core library `feniqs_lib` uses plugin and environment managers to load each emulator in an isolated virtual environment, avoiding conflicts and preserving reproducibility. All plugins conform to a unified interface for circuit execution, enabling fair comparisons across different emulators. This modular architecture also allows users to easily switch between simulation methods, versions or hardware platforms.

Configuration and Pre-processing: Before benchmarking, users can modify three distinct configuration files, in YAML format, to define FENIQS setup: `venv_deps.yaml` defines the environment, ●●●

Table 1. Supported frameworks within FENIQS, detailing their primary simulation methods, API languages, open-source status, and CPU/GPU hardware targets.

Framework	Method	API Language	Open Source	CPU	GPU
Cirq [12]	StateVector	Python	✓	✓	✓
Qiskit [13]	StateVector	Python	✓	✓	✓
	MPS	Python	✓	✓	✓
MIMIQ	StateVector	Julia		✓	
	MPS	Julia		✓	
PennyLane [14]	StateVector	Python	✓	✓	✓
Qulacs [15]	StateVector	Python	✓	✓	✓
Yao [16]	StateVector	Julia	✓	✓	✓
ProjectQ [17]	StateVector	Python	✓	✓	
QMatchTea [18]	MPS	Python	✓	✓	✓
MQT-DDS [19]	Decision Diagrams	Python	✓	✓	
Pyqrack [20]	Factorized Ket	Python	✓	✓	
Qrisp [21]	Sparse Matrix	Python	✓	✓	
Quimb [22]	MPS	Python	✓	✓	
	NT	Python	✓	✓	

●●● selecting emulator plugins and ensuring backend isolation; `config_backend.yaml` specifies emulator types and parameters; `optimizer.yaml` (optional) sets parameter ranges for MPS performance tuning. To address the strong impact of emulator-specific parameters (such as bond dimension for MPS, accuracy thresholds, or circuit pre-optimizations) on both run time and fidelity, FENIQS includes a noise-aware hyperparameter optimizer. It supports CMA-ES for single-objective optimization, minimizing run time while satisfying a fixed fidelity constraint, ideal when high accuracy is required. For broader trade-off exploration, NSGA-II and MOEA-D enable multi-objective optimization, generating a Pareto front that balances fidelity and run time. All candidate configurations are evaluated under run time noise, with optimized solution exportable to the backend YAML file for direct use in benchmarking.

Post-processing: FENIQS includes a set of Python scripts for parsing JSON result files and generating comparative plots. These tools facilitate the analysis of benchmark metrics such as run time, fidelity, and scalability across different emulators.

Metrics

FENIQS evaluates performance based on two direct core metrics: *run time* and *mirror circuit fidelity*.

The *run time* metric in FENIQS refers to the total wall-clock duration required to execute a quantum circuit. This includes all necessary operational steps, including circuit parsing from OpenQASM 2.0 format, quantum state initialization, circuit execution, and sampling of measurement outcomes. To account for system warm-up effects and inherent shot-to-shot variability in execution times, each benchmark circuit is executed multiple times, as specified by the users, and FENIQS reports the minimum run time (in seconds) achieved.

For assessing the accuracy of emulators, particularly those not based on statevector methods where direct fidelity calculation with a reference state can be intractable, FENIQS implements the *mirror-circuit fidelity*. This method evaluates an emulator's accuracy without relying on its internal fidelity estimates. It involves executing a circuit followed immediately by its inverse (the mirror circuit), and then measuring the probability of the system returning to its initial computational basis state. To prevent emulators from trivially optimizing the combined circuit to an identity operation, FENIQS supports the insertion of barrier operations after the forward circuit (where the emulator supports them).

Beyond presented primary metrics, FENIQS introduces indirect metric – scalability, which quantifies an emulator's ability to efficiently handle increasing problem size without loss of accuracy or excessive run time. This metric is assessed by analyzing trends in the primary metrics – run time and fidelity – collected by FENIQS across multiple circuit instances of varying qubit numbers for a given algorithm or circuit class. Typically, this involves evaluating how the measured run time $T_E(n)$ for the emulator E depends on the number of qubits n , subject to maintaining the mirror-circuit fidelity $F_E \geq 0.99$, and respecting an execution timeout of 300 seconds. Emulators exhibiting polynomial scaling of run time with n are generally considered more scalable for that class of circuits.

Results and Main Features

FENIQS robustness and versatility were demonstrated in a large-scale benchmarking work presented in (Leonteva et al. 2025), which assessed seven cutting-edge quantum emulators across diverse simulation techniques. This study included four MPS-based tools (Qiskit-MPS, Quimb-MPS, QMatchaTea-MPS, MIMIQ-MPS), as well as Quimb-TN, Pyqrack, and MQT-DDS. The benchmark suite featured 13 scalable circuits from the MQTBench library ², ranging

² <https://github.com/cda-tum/MQTBench>

from 4 to 1,024 qubits on CPU hardware. To present these results, Figure 2 reproduces a radar chart from (Leonteva et al. 2025), showing the maximum qubit counts achieved by each emulator across the 13 benchmark circuits. The radial axis uses a logarithmic scale, from 12 (innermost circle) to 1,024 qubits (outermost), providing a clear visual summary of scalability. Larger polygon areas indicate emulators capable of solving a greater fraction of circuits at higher qubit counts, highlighting their performance under utility-scale conditions.

Beyond its demonstrated use case, FENIQS offers several key features that make it a powerful tool for diverse benchmarking needs.

Flexibility of the Benchmarking Process: FENIQS enables highly customizable benchmarking by allowing users to group emulators based on simulation method (e.g., statevector, tensor network), hardware platform (CPU/GPU), or criteria such as numerical precision or run time limits. YAML-based configurations let users compare backends, versions, and execution modes without writing code. For example, the left panel of Figure 3 shows run time scaling of the “Amplitude Estimation” circuit across seven emulators (mentioned above) with increasing qubit counts on CPU; the right panel compares GPU-implemented statevector simulators, Qiskit and PennyLane, executing the *hdyn* task (simulating XYZ-Heisenberg dynamics).

Extensibility: FENIQS supports the addition of new emulators and optimization strategies with minimal core changes. Benchmark suites can be extended by adding OpenQASM 2.0 circuits, and JSON-formatted results integrate easily with external tools for advanced post-processing and visualization, without altering existing workflows.

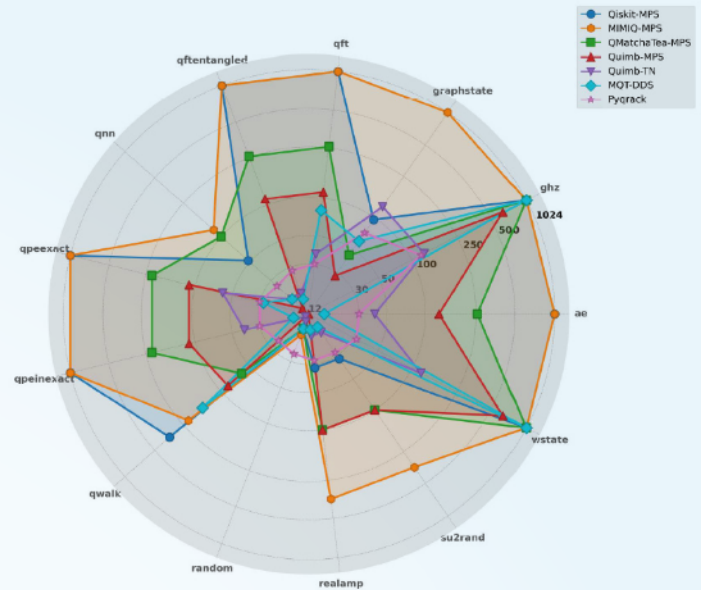


Figure 2: FENIQS output visualizing comparative emulator scalability (Leonteva et al. 2025). The radar chart displays the maximum achievable qubit counts for seven emulators across thirteen benchmark circuits. Each radial axis represents a specific benchmark algorithm, with concentric circles indicating the qubit scale from 12 to 1,024 (logarithmic). Larger polygon areas generally indicate superior overall scalability across the evaluated benchmark suite.

Efficiency and Accessibility: FENIQS streamlines benchmarking by automating setup, execution, and monitoring, reducing workflow time from hours to minutes. YAML configurations replace complex coding, making it accessible to users with less extensive programming and quantum emulators expertise.

Reproducibility and Clarity: YAML-based configurations and structured JSON results ensure reproducibility and simplify comparison across emulators.

...

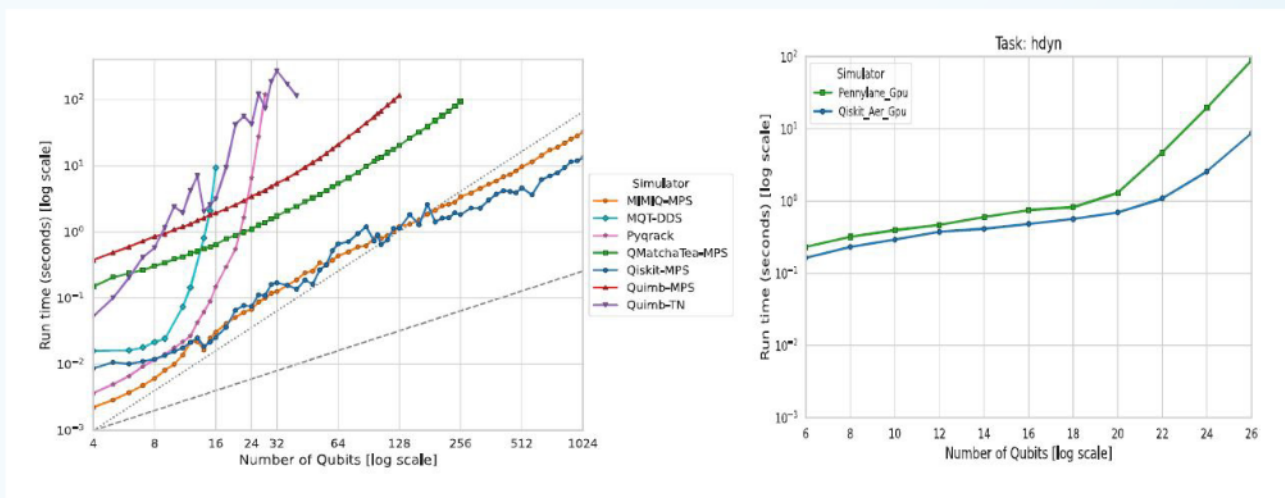


Figure 3: Run time plots generated by FENIQS, comparing emulator performance as a function of qubit count for two distinct quantum tasks and emulator sets. Both panels present run time in seconds versus the number of qubits. Left panel illustrates run time (in log scales) scaling for the amplitude-estimation benchmark circuit (Leonteva et al. 2025) for seven different emulators, as detailed in the legend. Dashed and dotted lines provide visual references for linear and quadratic scaling behaviors, respectively. Right panel shows a performance comparison of two statevector GPU-based simulators (PennyLane and Qiskit) on the *hdyn* benchmarks.

... Conclusions

We introduced FENIQS, a user-centric, problem-agnostic, and plugin-based framework for benchmarking quantum emulators. FENIQS streamlines comparative emulator studies through a unified API supporting twelve distinct backends, a modular plugin system, automated hyperparameter tuning, and robust performance metrics. It automates key benchmarking steps, saving users time and reducing the need for specialized expertise, by supporting the full benchmarking life-cycle, from scenario setup to result visualization, and enabling custom experiments. The practical utility of FENIQS in generating reliable performance insights, such as detailed emulator scalability profiles and assessments of classical simulation boundaries — as demonstrated in a comprehensive utility-scale benchmarking study (Leonteva et al. 2025). FENIQS thus serves as a valuable tool for the quantum computing community, aiding in emulator selection and fostering the advancement of simulation techniques. Future work includes extending support to quantum hardware to address execution variability and cost, and integrating machine-

learning-based accelerators for parameter tuning. The full source code is openly available on GitHub ³. ■

³ https://github.com/qperfect-io/feniqs_lite

Abstract

Classical quantum emulators are key tools for algorithm design, error correction, and noise mitigation. They rely on a variety of simulation methods such as statevector, tensor networks, matrix product states, decision diagrams and others. Their performance depends not only on simulation method and circuit properties, but also on emulator-specific parameters, pre-processing and implementation. Systematic benchmarking of emulators is essential for advancing simulation techniques and selecting the most suitable emulator for a given task. To address this need, we present an open-source, user-centric, problem-agnostic framework with a unified API for twelve leading emulators, delivering detailed performance insights for quantum computing research and development. ■

Références

- Arute, Frank, Kunal Arya, Ryan Babbush, et al. 2019. "Quantum Supremacy Using a Programmable Superconducting Processor." *Nature* 574 (7779): 505–10.
- Bergholm, Ville, Josh Izaac, Maria Schuld, et al. 2018. "PennyLane: Automatic Differentiation of Hybrid Quantum-Classical Computations." *arXiv Preprint arXiv:1811.04968*.
- Bowles, Joseph, Shah Nawaz Ahmed, and Maria Schuld. 2024. "Better Than Classical? The Subtle Art of Benchmarking Quantum Machine Learning Models." *arXiv Preprint arXiv:2403.07059*.
- Cirq Developers. 2025. Cirq (V1.5.0). Zenodo. <https://doi.org/10.5281/zenodo.15191735>.
- Cross, Andrew W, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. 2019. "Validating Quantum Computers Using Randomized Model Circuits." *Physical Review A* 100 (3): 032328.
- Finžgar, Jernej Rudi, Philipp Ross, Leonhard Hölcher, Johannes Klepsch, and Andre Luckow. 2022. "Quark: A Framework for Quantum Computing Application Benchmarking." 2022 IEEE International Conference on Quantum Computing and Engineering (QCE), 226–37.
- Gray, Johnnie. 2018. "Quimb: A Python Library for Quantum Information and Many-Body Calculations." *Journal of Open Source Software* 3 (29): 819. <https://doi.org/10.21105/joss.00819>.
- Jamadagni, Amit, Andreas M Läuchli, and Cornelius Hempel. 2024. "Benchmarking Quantum Computer Simulation Software Packages: State Vector Simulators." *Preprint arXiv:2401.09076*.
- Kim, Youngseok, Andrew Eddins, Sajant Anand, et al. 2023. "Evidence for the Utility of Quantum Computing Before Fault Tolerance." *Nature* 618 (7965): 500–505.
- King, Andrew D., Alberto Nocera, Marek M. Rams, Jacek Dziarmaga, Roeland Wiersema, et al. 2025. "Beyond-Classical Computation in Quantum Simulation." *Science* 388 (6743): 199–204.
- Leonteva, Anna, Guido Masella, Maxime Outteryck, Asier Piñeiro Orioli, and Shannon Whitlock. 2025. Comparative Benchmarking of Utility-Scale Quantum Emulators. <https://arxiv.org/abs/2504.14027>.
- Lubinski, Thomas, Joshua J Goings, Karl Mayer, et al. 2024. "Quantum Algorithm Exploration Using Application-Oriented Performance Benchmarks." *Preprint arXiv:2402.08985*.
- Montangero, Simone. 2018. *Introduction to Tensor Network Methods*. Springer. <https://doi.org/10.1007/978-3-030-01409-4>.
- Morvan, Alexis, B Villalonga, X Mi, et al. 2024. "Phase Transitions in Random Circuit Sampling." *Nature* 634 (8033): 328–33.
- Nation, Paul D, Abdullah Ash Saki, Sebastian Brandhofer, et al. 2025. "Benchmarking the Performance of Quantum Computing Software for Quantum Circuit Creation, Manipulation and Compilation." *Nature Computational Science*, 1–9.
- Seidel, Raphael, Sebastian Bock, René Zander, et al. 2024. Qrisp: A Framework for Compilable High-Level Programming of Gate-Based Quantum Computers. <https://arxiv.org/abs/2406.14792>.
- Steiger, Damian S, Thomas Häner, and Matthias Troyer. 2018. "ProjectQ: An Open Source Software Framework for Quantum Computing." *Quantum* 2: 49.
- Strano, Daniel, Benn Bollay, Aryan Blaauw, Nathan Shammah, William J. Zeng, and Andrea Mari. 2023. "Exact and Approximate Simulation of Large Quantum Circuits on a Single GPU." *Preprint arXiv:2304.14969*.
- Suzuki, Yasunari, Yoshiaki Kawase, Yuya Masumura, et al. 2021. "Qulacs: A Fast and Versatile Quantum Circuit Simulator for Research Purpose." *Quantum* 5: 559.
- Treinish, Matthew, Luciano Bello, Jake Lishman, et al. 2025. Qiskit/Qiskit: Qiskit 2.0.0 (2.0.0). Zenodo. <https://doi.org/10.5281/zenodo.15116124>.
- Wille, Robert, Stefan Hillmich, and Lukas Burgholzer. 2022. "Decision Diagrams for Quantum Computing." In *Design Automation of Quantum Computers*. Springer.