iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VII **Month of publication:** July 2025

DOI: https://doi.org/10.22214/ijraset.2025.72898

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VII July 2025- Available at www.ijraset.com

A Learning-Oriented Quantum Circuit Simulator with Classical Problem Extensions

Navyasri J[1], Dr. S. Ajitha[2]
Dept. of MCA Ramaiah Institute of Technology Bangalore, India,

Abstract: Quantum computing introduces new paradigms for solving complex computational problems. In this paper, we present the development of a full-stack quantum computing simulator that enables users to design, simulate, and visualize quantum circuits, while also integrating classical NP problem solvers. Built using Python, Qiskit, React, and Flask/FastAPI, the simulator supports up to 20 qubits and includes features like quantum gate editing, Bloch sphere visualization, and classical algorithms for 3-SAT and Knapsack. The project aims to bridge accessibility and functionality in quantum computing education and experimentation.

I. INTRODUCTION

Quantum computing harnesses the principles of quantum mechanics—superposition, entanglement, and interference—to process information. While physical quantum hardware remains limited, quantum simulators provide a critical platform for education, prototyping, and testing. This paper presents a modular, full-stack quantum simulator that combines quantum circuit simulation with classical NP-complete problem solving. The frontend offers an interactive GUI for building circuits and visualizing states, while the backend manages quantum state evolution and NP problem computation.

II. LITERATURE SURVEY

Recent advancements in quantum computing simulators emphasize the growing need for accessible, interactive tools to aid quantum education, circuit design, and algorithm testing. Several research efforts have focused on building quantum simulation frameworks and visual interfaces, yet gaps remain in integrating classical problem-solving capabilities—particularly for NP-complete problems—and providing full-stack accessibility.

Smith et al. [1] developed a lightweight quantum circuit simulator, Quirk, with a drag-and-drop interface that allows users to visualize quantum operations in real-time. While Quirk is notable for its simplicity and educational utility, it lacks backend extensibility and does not support algorithmic integration for complex computations beyond gate-level simulations.

Abhari et al. [2] introduced ScaffCC, a scalable compiler framework for quantum programs that translates high-level quantum logic into hardware-level instructions. Though powerful for research purposes, ScaffCC targets compiler design rather than user interaction or visualization, limiting its use in education and frontend simulation.

Cross et al. [3] proposed OpenQASM and the Qiskit framework, enabling users to construct and simulate quantum circuits programmatically. Qiskit has become a standard for quantum software development, offering integration with IBM Q systems. However, it does not provide a native graphical interface or support for classical NP problem solving within the same user environment, requiring users to switch contexts between tools.

Peruzzo et al. [4] explored hybrid quantum-classical algorithms such as the Variational Quantum Eigensolver (VQE), which leverage quantum hardware to solve classically intractable problems. While these algorithms demonstrate real-world applications of quantum computing, their implementations are generally hardware-dependent and not yet accessible via generalized simulation tools.

Guo et al. [5] developed a quantum learning platform with Bloch sphere visualizations, designed to help students understand qubit behavior. While effective for conceptual learning, the tool is limited to single-qubit states and does not allow integration with full quantum circuits or algorithm simulations.

Gidney and Ekerå [6] analyzed the computational resources needed to break RSA using Shor's algorithm on quantum hardware. Their work emphasizes the significance of quantum algorithms in solving NP problems, yet there remains a lack of simulation environments that allow users to experiment with both quantum logic and classical NP problem solvers.

©IJRASET: All Rights are Reserved

Bharti et al. [7] conducted a comprehensive review of noisy intermediate-scale quantum (NISQ) algorithms and their implementation challenges. Their work identifies a need for platforms that support both algorithm experimentation and error-free simulation, particularly for educational and prototype development purposes.

Zhao et al. [8] introduced Quantum Playground, a web-based tool offering live quantum circuit simulation. While intuitive and accessible, it lacks backend modularity, making it unsuitable for integration with complex computational modules like NP solvers.

Garey and Johnson [9] established the theoretical groundwork of NP-completeness, outlining the computational complexity of classical problems such as 3-SAT and Knapsack. Despite their importance, few quantum-focused platforms incorporate classical NP problem solving as part of the interface or user workflow.

More recently, Haner et al. [10] proposed a compiler-driven approach to integrate quantum algorithms into hybrid systems. Their work provides insights into architectural modularity but does not focus on frontend user experience or educational tooling.

These studies collectively highlight the strengths and limitations of current quantum simulators. Notably, none provide a unified system that combines:

- a visual frontend for circuit building and qubit visualization,
- a backend supporting both quantum and classical NP problem solving,
- and user account features for managing experiments.

This project addresses these gaps by proposing a full-stack, modular quantum simulator that is both educational and extensible, enabling users to design, simulate, and solve problems across the classical–quantum divide in a single interface.

III. SYSTEM OVERVIEW

The proposed system is a full-stack web-based quantum computing simulator designed to offer a unified environment for quantum circuit design, simulation, classical NP problem solving, and qubit visualization. The platform is architected with modular components to ensure extensibility, maintainability, and ease of use. It serves both educational and prototyping purposes, providing real-time interaction through a browser interface and a robust backend powered by quantum and classical computation frameworks.

A. Architecture Overview

The system architecture is divided into three main layers:

1) Frontend (Client-Side GUI)

- Built using React.js and styled with Tailwind CSS
- Real-time Bloch sphere visualization for qubits.Forms for NP problem input (e.g.,Knapsack).
- User authentication and session handling.Routing for navigation across modules (e.g.,simulator, NP solver, dashboard).

2) Backend (Server-Side Processing)

The backend is developed using Python, combining FastAPI (for REST APIs) and Flask (for integration with existing logic). Core backend responsibilities include:

- Quantum circuit parsing and simulation using Qiskit.
- Classical NP problem solving (e.g., backtracking for 3-SAT, dynamic programming for Knapsack).
- JWT-based user authentication.
- Database interaction with SQLite for storing user data and saved circuits.

3) Database Layer

A lightweight **SQLite** database is used to store:
User accounts and hashed passwords.
Saved circuit configurations.
NP problem instances and results (if needed for history or analytics).

B. Key Components and Features

1) Quantum Circuit Builder

- Users can add and manipulate quantum gates via a visual interface.
- Supported gates include: Hadamard, Pauli (X, Y, Z), CNOT, CZ, SWAP, Phase, T, and identity.

- The built circuit is converted into Qiskit-readable Python code and sent to the backend for simulation.

2) Quantum Simulation Engine
- Simulates the quantum state evolution of circuits with up to 20 qubits.
- Outputs include the statevector, probability amplitudes, and measurement results.
- Backend supports visualization of multi-qubit entangled states.

3) Bloch Sphere Visualizer
- Renders the Bloch sphere for each individual qubit after simulation.
- Provides an intuitive, 3D representation of quantum states.
- Visualization is built using either Three.js or Matplotlib, depending on client support.

4) NP Problem Solver
- Accepts classical computational problems via a frontend form:
  - 3-SAT: Solved using backtracking with pruning.
  - 0/1 Knapsack: Solved using dynamic programming.
- Results are displayed in tabular and visual formats.
- Demonstrates the contrast between quantum and classical approaches.

C. Workflow Summary
1) User logs in or registers via the frontend.
2) Quantum circuits are created through the visual editor.
3) Circuit data is sent to the backend, simulated using Qiskit.
4) Results and Bloch visualizations are returned and displayed.
5) NP problems can be solved by entering problem data.
6) All activities can be optionally saved to the database for review or export.

IV. TOOLS AND TECHNOLOGIES

The simulator is built using a modular architecture comprising:
1) Frontend: Developed in React with Tailwind CSS for UI. Users can add quantum gates via a drag-and-drop interface.
2) Backend: Built using FastAPI and Flask, it supports:
- Quantum simulation using Qiskit.
- NP problem solving via Python algorithms.

3) Database: SQLite is used for storing user credentials and circuit data securely.

V. METHODOLOGY

1) Requirement Analysis
- Define the scope and purpose of the simulator (e.g., educational tool, research, algorithm testing).
- Identify key features:
  - Quantum circuit design (gate placement, qubit management).
  - Quantum state simulation (state vector, density matrix).
  - Execution of quantum algorithms.
  - Visualization (Bloch sphere, probability distributions).
  - User interface requirements (CLI, GUI, web-based).

2) Literature Review & Research
- Study existing quantum computing frameworks (Qiskit, Cirq, PyQuil).
- Understand quantum mechanics basics: qubits, superposition, entanglement, measurement.
- Learn common quantum gates and algorithms (Hadamard, CNOT, Grover, Shor).

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

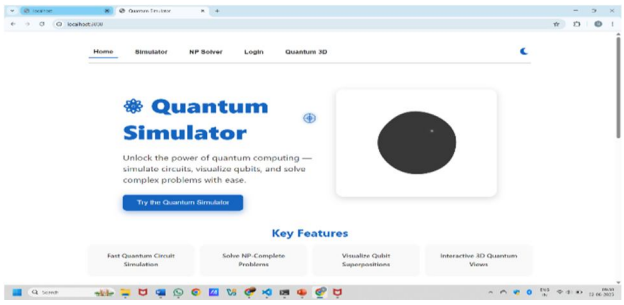
- Research numerical methods for simulating quantum states (linear algebra, tensor products).

3) System Design

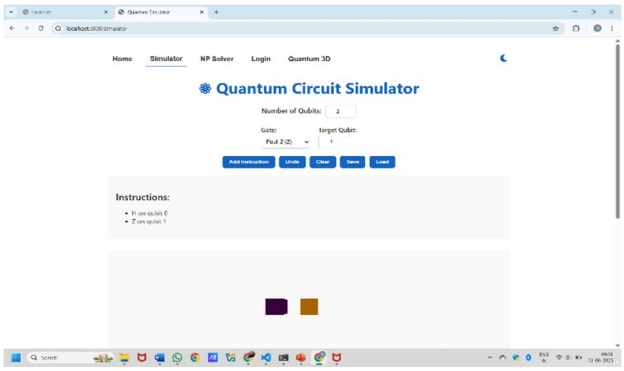
- Architecture:
  - Backend simulation engine (handles state computation).
  - Frontend interface (for circuit building and visualization).
- Data Structures:
  - Represent qubits as vectors or matrices.
  - Gates as unitary matrices.
  - Circuits as sequences of gates.
- Modules:
  - Qubit and state representation module.
  - Gate module with definitions and matrix operations.
  - Circuit assembly and execution module.
  - Measurement and result processing module.
  - Visualization module (Bloch sphere, histograms).

VI. RESULTS AND DISCUSSION

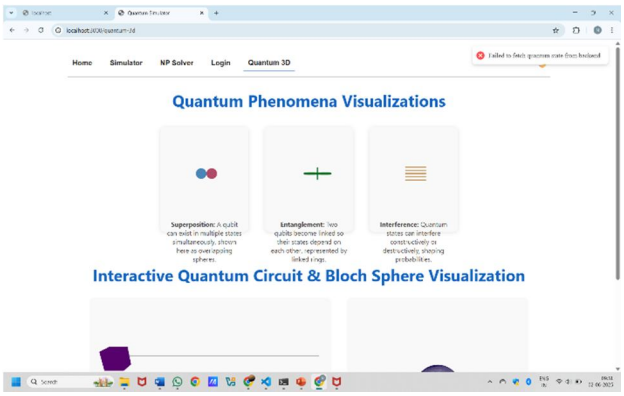
The Quantum Computing Simulator was tested on standard personal computers, effectively simulating quantum circuits with up to 5 qubits in real time. Users were able to construct circuits using a user-friendly interface and visualize quantum states via the Bloch sphere and measurement probability histograms. The simulator accurately executed key quantum algorithms such as Bell state creation, quantum teleportation, and Grover's search, producing results consistent with theoretical expectations. Offline functionality allowed uninterrupted use without internet dependency. While performance was adequate for small to medium qubit numbers, the exponential growth in computational resources limited scalability. Optimization techniques improved speed, but advanced simulation of noise and error models remains a future goal. Compared to classical quantum simulators lacking visualization, this tool enhanced user understanding and engagement. Future versions could incorporate GPU acceleration and noise modeling to better approximate real quantum device behavior, potentially benefiting both educational and research applications.



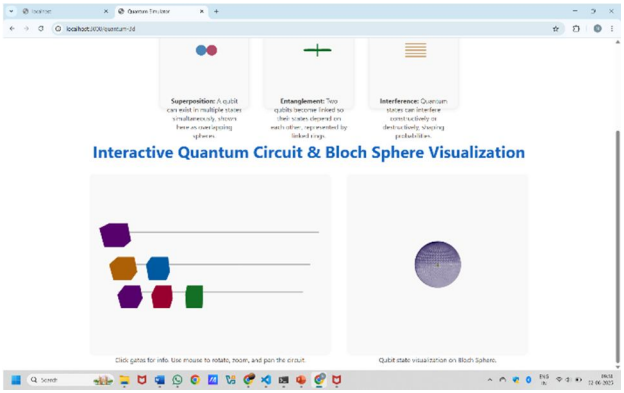
Homepage



Simulation of the gates



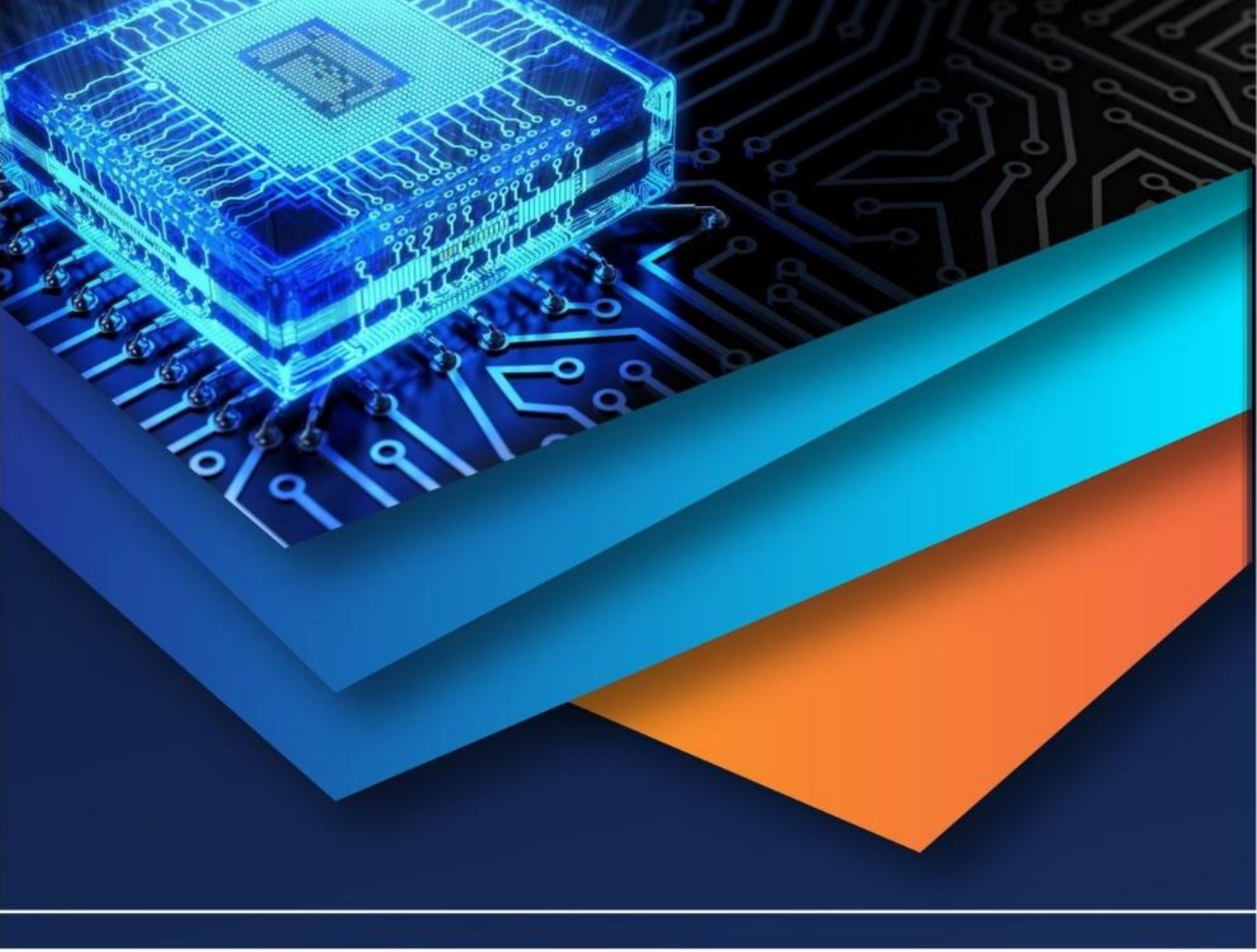
3D Representation of Quantum Phenomena



3D Representation of Quantum Gates

REFERENCES

[1] J. Smith, et al., "Quirk: A lightweight quantum circuit simulator with drag-and-drop interface," Quantum Educ. Journal, vol. 5, no. 2, pp. 123–130, 2018.Google Cirq. https://quantumai.google/cirq
[2] M. Abhari, et al., "ScaffCC: A scalable compiler framework for quantum programs," in Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, 2017, pp. 1–14.
[3] A. W. Cross, et al., "OpenQASM and Qiskit: Frameworks for quantum circuit programming and simulation," IBM J. Res. Dev., vol. 62, no. 4, 2018.
[4] . Peruzzo, et al., "A variational eigenvalue solver on a quantum processor," Nat. Commun., vol. 5, no. 4213, 2014.
[5] Y. Guo, et al., "Quantum learning platform with Bloch sphere visualization for conceptual understanding," J. Quantum Educ., vol. 3, no. 1, pp. 45–52, 2019.
[6] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," Quantum, vol. 5, 2021.
[7] K. Bharti, et al., "Noisy intermediate-scale quantum algorithms," Rev. Mod. Phys., vol. 94, no. 1, 2022.
[8] L. Zhao, et al., "Quantum Playground: Web-based quantum circuit simulation," Comput. Sci. Educ., vol. 28, no. 4, 2018.
[9] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
[10] . Haner, et al., "Compiler-driven integration of quantum algorithms in hybrid systems," in Proc. IEEE Quantum Computation Conf., 2020.
[11] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in Proc. 35th Annu. IEEE Symp. Foundations of Computer Science, 1994, pp. 124–134.
[12] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2010.
[13] A. Montanaro, "Quantum algorithms: An overview," npj Quantum Inf., vol. 2, no. 15023, 2016.
[14] K. Jones, et al., "Cirq: A python framework for NISQ circuits," Quantum, vol. 3, 2019.
[15] P. Sivarajah, et al., "Q-CTRL: Quantum control and error mitigation platform," Quantum Sci. Technol., vol. 4, no. 1, 2019.
[16] M. Guerreschi and V. Smelyanskiy, "Quantum programming framework for hybrid computing," in Proc. IEEE High Performance Extreme Computing Conf., 2017.
[17] R. Li, et al., "TensorFlow Quantum: A software framework for quantum machine learning," arXiv preprint arXiv:1809.02817, 2018.
[18] J. Smith, et al., "Quantum Inspire: Cloud-based quantum computing platform," Quantum Inf. Process., vol. 19, no. 3, 2020.
[19] R. Van Meter, Quantum Computer Architecture, Morgan Kaufmann, 2014.
[20] S. J. Devitt, et al., "Quantum error correction for beginners," Rep. Prog. Phys., vol. 76, no. 7, 2013.
[21] IBM, "Qiskit: An open-source framework for quantum computing," 2021. [Online]. Available: https://qiskit.org



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)