QuTiP-MRL: A Library for Multiple-Valued Reversible Logic Simulations

Fabio Pievani[1], Asma Taheri Monfared[1][0009−0001−0465−5579], Andrea Bombarda[1][0000−0003−4244−9319], and Angelo Gargantini[1][0000−0002−4035−0131]

University of Bergamo, Bergamo, Italy
f.pievani1@studenti.unibg.it,
{asma.taherimonfared,andrea.bombarda,angelo.gargantini}@unibg.it

Abstract. Reversible logic is a key technology for low-power and quantum computing, as it allows computation without information loss and minimizes energy dissipation. Meanwhile, multiple-valued logicoffers advantages such as reduced circuit complexity and improved data representation compared to binary systems. Combining the benefits of multiple-valued logic and reversibility opens new possibilities for efficient and scalable computing architectures. However, there has been a lack of tool support for designing and simulating multiple-valued reversible circuits. In this paper, we present QuTiP-MRL, a Python-based library for designing and analyzing multiple-valued reversible logic circuits, with current support for ternary and quaternary logic. Built on top of the QuTiP (Quantum Toolbox in Python) framework, QuTiP-MRL addresses this gap by providing a set of well-defined multiple-valued reversible gates and tools to construct and simulate multiple-valued circuits. This library offers an environment for exploring the behavior and properties of multiple-valued reversible systems, supporting research and development in emerging computational paradigms.

Keywords: Multiple-Valued Logic · Reversible Circuits · Circuit Simulation · Circuit Visualization.

1 Introduction

Reversible computation has become a key paradigm in modern computing, particularly in the domains of low-power design and quantum information processing. Unlike conventional logic circuits that lose information during computation, reversible logic ensures bijective transformations, allowing outputs to be traced back to their original inputs [1]. This information-preserving property is essential for quantum computing, where all operations must be unitary and reversible by nature [17]. Additionally, reversible logic minimizes energy dissipation, in accordance with Landauer's principle, which states that the erasure of a single bit of information results in an energy cost of at least $kT \ln 2$ joules [16].

In parallel, there has been growing interest in Multiple-Valued Logic (MVL), which extends binary logic to systems with more than two states. MVL, especially ternary and quaternary logic, offers several theoretical and practical advantages, such as reduced circuit complexity, lower interconnect overhead, and

increased data density [10,14,3]. In the context of quantum computing, these benefits are particularly compelling, as physical implementations of qudits (quantum digits with $d > 2$ levels) are becoming increasingly feasible on platforms such as trapped ions, photonic systems, and superconducting circuits. Ternary and quaternary quantum logic are also known to exhibit better noise resilience and fault tolerance compared to binary approaches [8].

Despite these advantages, the practical development, simulation, and analysis of multiple-valued reversible logic circuits remain a major challenge. Most existing quantum circuit design and simulation tools, including widely-used frameworks like Qiskit [11], Cirq [18] and QuTiP [12] focus exclusively on binary systems and qubit-based gates. Researchers interested in non-binary reversible logic have to resort to low-level mathematical modeling or custom simulations, which are error-prone and time-consuming. There is currently no well-supported library that enables users to design, simulate, and visualize ternary or quaternary reversible logic circuits in an accessible and modular way [20].

To address this gap, we introduce QuTiP-MRL, a Python-based simulation library for Multiple-Valued Reversible Logic (MRL), with current support for both ternary and quaternary logic systems. Built on top of the QuTiP (Quantum Toolbox in Python) framework, QuTiP-MRL enables users to define multi-valued reversible gates, build custom circuits, and simulate their behavior using quantum mechanical state vectors and operators. The library includes a growing set of built-in gates for ternary logic, along with utilities for constructing larger systems, visualizing circuit behavior, and analyzing quantum cost metrics.

QuTiP-MRL is designed to support both research and educational applications. It provides a high-level, modular API for simulating multiple-valued reversible logic, and it is open-source to encourage collaboration and extensibility. By making the simulation of multiple-valued reversible circuits more accessible, this library aims to accelerate the exploration of emerging computational paradigms that go beyond the limitations of binary logic.

This paper is structured as follows: Section 2 describes the library implementation, including the overall structure, supported gate sets, and the functionalities offered by QuTiP-MRL. Section 3 provides a detailed demonstration of the library. Section 4 discusses related works and comparison. Finally, the conclusion of this work is given in Section 5.

2 Library implementation

In this section, we describe our QuTiP-MRL library and the operations it supports. It is based on QuTip [15] and it is available at `https://github.com/foselab/QuTiP-MRL`. To minimize the effort required for users familiar with Qiskit [11], we designed QuTiP-MRL with an interface that closely resembles that of Qiskit and with similar visualization modes.

The main class of this library is QuditCircuit, which provides all the core functionalities for gate addition, circuit simulation, and visualization in a unified interface. The library is qutrit ready, therefore it includes the matrices used

by ternary reversible logic gates and their specific visualization, but it also works with general qudits through user-specified matrices, making QuTiP-MRL compatible with any basis states qudits. In the following, we describe QuTiP-MRL functionalities in terms of circuit construction (Section 2.1), simulation backends (Section 2.2), and visualization (Section 2.3).

2.1 Internal Circuit Structure

The first step the user must perform when using QuTiP-MRL is the circuit initialization through the QuditCircuit class. When instantiating a QuditCircuit object, users must specify the number of qudits in the modeled circuit and, possibly, the number of states. By default, QuTiP-MRL works with qutrits, thus it assumes working with ternary logic. For example, the call q = QuditCircuit(4) creates a ternary circuit with 4 qutrits, while q = QuditCircuit(3, 4) creates a quaternary circuit with 3 qudits, each with 4 states.

When an instance q of a QuditCircuit is available, users can add quantum gates. QuTiP-MRL supports ternary *shift* gates, ternary *Muthukrishnan–Stroud* gates, and *custom* gates, in the case in which more than 3 states or different gates are required.

Concerning *shift* gates, we provide functions for all ternary gates (id, plus1, plus2, one_two, zero_one, zero_two). All of them require as input parameter the index of the target qutrit. For example, q.plus2(0) adds a +2 gate to the qutrit 0. Concerning controlled gates, i.e., *Muthukrishnan–Stroud* gates, QuTiP-MRL provides functions for all quantum multiple valued gates (i.e., c_plus1, c_plus2, c_one_two, c_zero_one, c_zero_two). Similarly to shift gates, controlled gates require a parameter indicating the target qudit and, in addition, they require the index of the control qudit. For example, The command q.c_plus2(3,0) represents a controlled +2 gate, where the first argument (3) is the control qudit and the second argument (0) is the target qudit.

Finally, when working with quaternary logic, more than 4 states, or not available gates are needed, two additional methods are provided: custom_gate and c_custom_gate. These two functions allow for defining alternative shift or controlled gates. Both functions follow the same structure as those for ternary gates. Additionally, users must input a matrix in the form of numpy.array and optionally the name of the gate for visualization purposes. For example, the command q.c_custom_gate(np.array([...]), 3,0,"CG1") adds a new gate, corresponding to the matrix specified as the first parameter to the qudit 0, controlled by the qudit 3. The gate is called, for visualization purposes,"CG1".

2.2 Simulation Backends

QuTiP-MRL allows users to simulate MVL circuits by using two backends, namely the one based on *full matrix* simulation and the one *einsum-based*.

Once a QuditCircuit instance q is available, the full matrix simulation can be performed by using the q.simulate_fullmatrix() command. QuTiP-MRL initializes the system in the $|0\rangle$ state for each qudit, applies each gate to the system state,

and calculates and prints the density matrix for each individual qudit, as well as the probability for each possible final measurement. During this operation, the system state is evolved step by step by performing matrix multiplication, for each gate, on the full quantum state. As a result, the data size grows rapidly, and simulation becomes computationally expensive: The bigger the number of states, the less qudits can be used in the circuit. For example, when using ternary it is recommended to limit the circuit to a maximum of 8 qutrits. Note that, when using this simulation mode, unlike other available libraries, QuTiP-MRL supports superposition [5].

Similarly, for the einsum-based simulation [19], once a QuditCircuit instance q is available, it can be performed by using the q.simulate_einsum() command. This method evolves the state of the circuit with tensor contractions using NumPy's einsum, which performs string manipulation, with index-wise operations. It allows the simulation of circuits with a relatively large number of qudits, as it avoids explicit matrix multiplication, which depends on the number of states of the qudit. For example, in ternary logic, 17 qutrits can be used with this kind of simulation. The quantum state is represented as a multidimensional tensor with shape $[d_1, d_2, \ldots, d_m]$ where d is the number of states and m is the number of qudits. The main limitation of this approach is that it only provides the marginal probability distributions of each qudit, losing quantum information in the process.

An example of the output obtained with each of the simulation backends will be shown in Section 3.3.

2.3 Rendering and Visualization

Rendering and visualizing logic circuits is of paramount important, as it allows for visually checking whether the circuit has been designed correctly and all gates have been connected to the correct qudits.

QuTiP-MRL offers, for circuit rendering, the same interface as Qiskit. Once a QuditCircuit instance q is available, multiple-valued circuits can be visualized by calling the q.draw() method. It offers two different visualization methods: ASCII and Matplotlib-based. In ASCII mode the circuit is displayed as plain text, making it suitable for quick inspection in command-line interfaces or automatic post-processing. In Matplotlib mode [2], the circuit is rendered as a structured and color-coded diagram, allowing for a clearer visualization of gate placement, control lines, and circuit depth. The former is triggered when no parameter is passed to the draw method, while the latter is triggered by passing 'mpl'. Note that the Matplotlib library is required to use this functionality.

Regardless of the chosen visualization technique, users may want to visually separate input preparation gates from the rest of the circuit. For this reason, the QuTiP-MRL library provides the q.barrier() method to draw input barriers.

An example of mpl visualization is reported and discussed in Section 3.2.

3 Demonstration

To showcase the capabilities of QuTiP-MRL, we present the construction and simulation of a reversible ternary full adder circuit [8]. This example highlights key features of the library, including gate-based circuit design, visualization modes, and simulation backends. We emphasize that, while this example focuses on a ternary logic circuit, QuTiP-MRL is designed to support reversible circuits in both ternary and quaternary logic. Further examples can be found in our GitHub repository at https://github.com/foselab/QuTiP-MRL/tree/main/examples.

3.1 Circuit Design

We report the Python code designing the circuit in Listing 1.1. Initially, we instantiate a circuit with four qutrits. Since we do not specify the number of possible dimension values, qutrits are used by default. The first two qutrits (q_0 and q_1) represent the ternary inputs; After circuit simulation, q_0 will contain the final sum, while q_2 and q_3 will be used to store the carry-in and the carry-out, respectively.

To initialize the inputs, we apply shift gates. The command `plus2(0)` applies a cyclic increment by two modulo 3 to the state of q_0. This operation shifts the current state by two. Similarly, `plus2(1)` applies the same shift to q_1. These are 1-qutrit shift (uncontrolled) gates that modify the qutrit's value.

After setting the initial inputs, we use the barrier() method to insert a vertical separator in the circuit. This barrier acts as a visual and logical boundary between the input initialization and the main computation. It does not affect the circuit's functionality but improves readability in both ASCII and graphical visualizations.

Muthukrishnan and Stroud gates (Controlled Shift gates) are used for conditional logic. For instance, `c_plus2(1, 0)` applies a +2 shift to q_0 only if q_1 is in the $|2\rangle$ state. In this command, the first argument (1) refers to the *control* qutrit, and the second (0) is the *target* qutrit. Another example is `c_one_two(0, 1)`, which swaps the $|1\rangle$ and $|2\rangle$ states of q_1, but only if q_0 is in state $|2\rangle$. These controlled operations are crucial for expressing reversible conditional logic.

The commands reported in Listing 1.1 implement the full adder logic using only reversible ternary gates, ensuring no information loss during computation.

3.2 Visualization

The code we use to visualize the circuit, in the two available modes, is reported in Listing 1.2. Figure 1 reports the Matplotlib-based visualization of the full adder.

```
full_adder = QuditCircuit(4)
# Set initial input states
full_adder.plus2(0) # Q0 = |2>
full_adder.plus2(1) # Q1 = |2>
full_adder.barrier()
# Full adder logic
full_adder.c_plus2(1, 0)
full_adder.c_one_two(0, 1)
full_adder.c_plus1(1, 3)
full_adder.c_one_two(0, 1)
full_adder.plus1(1)
full_adder.c_plus1(1, 0)
full_adder.plus2(1)
full_adder.c_plus2(2, 0)
full_adder.c_one_two(0, 2)
full_adder.c_plus1(2, 3)
full_adder.c_one_two(0, 2)
full_adder.plus1(2)
full_adder.c_plus1(2, 0)
full_adder.plus2(2)
```

Listing 1.1. Python code describing the implementation of a reversible ternary full adder with QuTiP-MRL

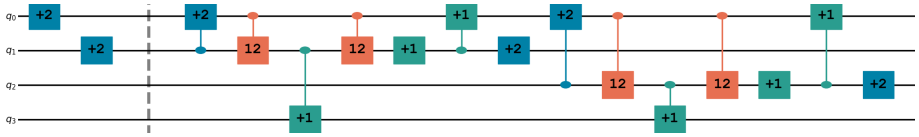

Fig. 1. Matplotlib-based visualization of the ternary full adder circuit.

3.3 Simulation

As described in Section 2, QuTiP-MRL provides two simulation engines [13] optimized for different circuit sizes. In our demonstration example, both modes are suitable as we only have four qutrits. The two modes can be invoked as in Listing 1.5. Both simulation modes start from the $|0000\rangle$ state and apply gates sequentially. The full matrix method outputs individual density matrices, while the einsum method provides final-state probabilities. More specifically, the full matrix method provides detailed information about each qutrit state in the form of 3×3 density matrices as well as the probabilities for each final possible measure, as shown in Listing 1.3, while the einsum-based simulation produces the state probabilities in the computational basis, as reported in Listing 1.4.

4 Related works and comparison

The increasing complexity of quantum algorithms and architectures has driven the development of a variety of libraries and frameworks for the design, simulation, and visualization of quantum circuits [9]. These tools aim to facilitate efficient prototyping, correctness verification, and performance optimization across diverse quantum computing models.

```
full_adder.draw() # ASCII representation
full_adder.draw('mpl') # Matplotlib visualization
```

Listing 1.2. Python code for the visualization of a circuit with QuTiP-MRL

```
Qudit 0 Density Matrix:
[[0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j]]
Qudit 1 Density Matrix: [...]
Final measurement probabilities:
|1201>: 100.00%
```

Listing 1.3. Full matrix simulation

```
Qudit 0 state probabilities:
  0
  1
  0

Qudit 1 state probabilities:
[...]
```

Listing 1.4. Einsum-based simulation

```
full_adder.simulate_fullmatrix() # Prints qutrit density matrices
full_adder.simulate_einsum() # Prints qutrit state probabilities
```

Listing 1.5. QuTiP-MRL simulation of the quantum ternary full adder

When it comes to quantum circuits, the most used library is Qiskit [11], developed by IBM. Similarly, Quantum++ is a high-performance C++11 library that supports simulation of arbitrary quantum processes, including classical reversible logic operations, making it suitable for mixed classical-quantum circuits [6]. Qibo [4] is a Python-based framework providing hardware-accelerated simulation capabilities with usability across CPUs, GPUs, and multi-GPU systems. However, all these libraries do not have a proper means to deal with multiple-valued circuits and with their simulation.

Lambert et al. [15] introduced QuTiP, an open-source software designed for simulating the dynamics of open quantum systems. Although it offers a robust simulation framework, its logic and syntax differ significantly from Qiskit, which is more familiar to most users. To address this, we present QuTiP-MRL in this work—a tool that enhances QuTiP by offering a more user-friendly interface and circuit visualization similar to that of Qiskit. In [7], the authors extended the Cirq [18] open-source framework from Google to support multiple-value logic. Similar to QuTiP, its visualization capabilities lack in a proper graphical representation, such as the one we support in QuTiP-MRL. Moreover, the tool proposed in [7] supports circuits of up to 14 qutrits, while QuTiP-MRL allows users to work with 17 qutrits. Additionally, the existing tool does not support superposition, while our tool does.

To the best of our knowledge, despite the recent advancements in the quantum circuits field, few frameworks directly support multi-valued quantum circuits. Moreover, none of these frameworks offers simulation capabilities for circuits of the same size as those supported by QuTiP-MRL or provides visualization similar to the well-known Qiskit's style and the one allowing the simulation of bigger circuits.

5 Conclusion

This work addresses the current lack of tools for quantum multi-valued logic circuits by introducing a comprehensive environment for their design, simula-

tion, and visualization—an essential step toward enabling more effective circuit development and supporting critical activities such as validation and analysis. In this paper, we have presented QuTiP-MRL, a Python library for the design, simulation, and visualization of multi-valued reversible quantum circuits. It is, to the best of our knowledge, the only available tool supporting the design, simulation, and visualization of multiple-valued quantum circuits by keeping the same structure and logic of Qiskit. We have demonstrated how to use QuTiP-MRL with a simple ternary full-adder circuit, starting from its design, to the simulation and visualization. Future improvements could include further code optimization allowing users to deal with even bigger and more complex circuits.

Acknowledgments. This work has been partially funded by project ANTHEM (AdvaNced Technologies for Human-centrEd Medicine) - Grant PNC0000003 – CUP: B53C22006700001 and by project PRIN 2022 SAFEST (Trust assurance of Digital Twins for medical cyber-physical systems), funded by the EU - NGEU, Mission 4, Component 2, Investment 1.1, CUP F53D23004230006, under the National Recovery and Resilience Plan (NRRP).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bennett, C.H., Shor, P.W.: Quantum information theory. IEEE transactions on information theory **44**(6), 2724–2742 (1998). https://doi.org/10.1109/18.720553
2. Bisong, E.: Matplotlib and Seaborn, pp. 151–165. Apress, Berkeley, CA (2019). https://doi.org/10.1007/978-1-4842-4470-8_12
3. Dong, H., Lu, D., Li, C.: A novel qutrit representation of quantum image. Quantum Information Processing **21**(3), 108 (2022). https://doi.org/10.1007/s11128-022-03450-8
4. Efthymiou, S., Ramos-Calderer, S., et al.: Qibo: a framework for quantum simulation with hardware acceleration. Quantum Science and Technology **7**(1), 015018 (dec 2021). https://doi.org/10.1088/2058-9565/ac39f5
5. Friedman, J.R., Patel, V., et al.: Quantum superposition of distinct macroscopic states. Nature **406**(6791), 43–46 (Jul 2000). https://doi.org/10.1038/35017505
6. Gheorghiu, V.: Quantum++: A modern c++ quantum computing library. PLOS ONE **13**(12), e0208073 (Dec 2018). https://doi.org/10.1371/journal.pone.0208073
7. Gokhale, P., Baker, J.M., et al.: Asymptotic improvements to quantum circuits via qutrits. In: Proceedings of the 46th International Symposium on Computer Architecture. p. 554–566. ISCA '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3307650.3322253
8. Haghparast, M., Wille, R., Monfared, A.T.: Towards quantum reversible ternary coded decimal adder. Quantum Information Processing **16**, 1–25 (2017). https://doi.org/10.1007/s11128-017-1735-3
9. Hamid, M., Alam, B., Pal, O.: Comparative study of quantum computing tools and frameworks. In: Innovation and Emerging Trends in Computing and Information Technologies. pp. 87–104. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-80842-5_8

10. Hurst: Multiple-valued logic—its status and its future. IEEE transactions on Computers **100**(12), 1160–1179 (1984). https://doi.org/10.1109/TC.1984.1676392
11. Javadi-Abhari, A., Treinish, M., Krsulich, K., et al.: Quantum computing with qiskit (2024). https://doi.org/10.48550/ARXIV.2405.08810
12. Johansson, J.R., Nation, P.D., Nori, F.: Qutip: An open-source python framework for the dynamics of open quantum systems. Computer physics communications **183**(8), 1760–1772 (2012). https://doi.org/10.1016/j.cpc.2012.02.021
13. Jozsa, R.: On the simulation of quantum circuits. https://doi.org/10.48550/ARXIV.QUANT-PH/0603163
14. Klimov, A., Guzmán, R., et al.: Qutrit quantum computer with trapped ions. Physical Review A **67**(6), 062313 (2003). https://doi.org/10.1103/PhysRevA.67.062313
15. Lambert, N., Giguère, E., et al.: Qutip 5: The quantum toolbox in python (2024). https://doi.org/10.48550/ARXIV.2412.04705
16. Landauer, R.: Irreversibility and heat generation in the computing process. IBM journal of research and development **5**(3), 183–191 (1961). https://doi.org/10.1147/rd.53.0183
17. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge university press (2010). https://doi.org/10.1017/CBO9780511976667
18. Omole, V., Tyagi, A., et al.: Cirq: A python framework for creating, editing, and invoking quantum circuits (2020), https://github.com/quantumlib/Cirq
19. Pan, F., Gu, H., Kuang, L., Liu, B., Zhang, P.: Efficient quantum circuit simulation by tensor network methods on modern gpus. ACM Transactions on Quantum Computing **5**(4) (Nov 2024). https://doi.org/10.1145/3696465
20. Taheri Monfared, A., Ciriani, V., Haghparast, M.: Qutrit representation of quantum images: new quantum ternary circuit design. Quantum Information Processing **23**(8), 288 (2024). https://doi.org/10.1007/s11128-024-04484-w