Comparison of Quantum Circuit Simulation Methods

Ibraheem AlYousef
Supervisor: Dr. Alawi AlSaqqaf
King Fahd University of Petroleum and Minerals

December 10, 2025

Abstract

We compare five quantum circuit simulation methods—state-vector, density matrix, matrix product state (MPS), unitary, and pulse-level—by executing Grover's search algorithm across distinct simulation backends. The comparison examines trade-offs between accuracy, computational cost, memory scaling, and appropriate use cases for each method. All methods achieve the expected theoretical success probability, with state-vector simulation offering the best memory efficiency for pure states, while pulse-level simulation via QuTiP provides insight into hardware-level Hamiltonian dynamics.

Contents

1 Introduction

Quantum circuit simulators employ various mathematical representations to model quantum state evolution. Each representation has different memory requirements, computational complexity, and applicability to specific problems. We investigate:

- How do state-vector, density matrix, MPS, unitary, and pulse-level simulations differ in their approach?
- What are the memory and time scaling behaviors of each method?
- How can we verify that simulations produce correct results?
- When should each method be used in practice?

2 Background

2.1 Grover's Search Algorithm

Grover's algorithm provides a quadratic speedup for searching an unsorted database of $N = 2^n$ items. Given an oracle that marks a target state $|t\rangle$, the algorithm amplifies the probability of measuring $|t\rangle$ from $1/N$ to nearly 1 using $O(\sqrt{N})$ oracle queries.

The algorithm consists of two main operations applied iteratively:

1. **Oracle:** Flips the phase of the target state: $|t\rangle \to -|t\rangle$
2. **Diffuser:** Reflects all amplitudes about their mean, amplifying the marked state

After k iterations, the probability of measuring the target state is:

$$P_{\text{target}} = \sin^2((2k+1)\theta), \quad \text{where } \theta = \arcsin(1/\sqrt{N}) \tag{1}$$

The optimal number of iterations is $k_{\text{opt}} = \lfloor \pi\sqrt{N}/4 \rfloor$, achieving success probability $> 96\%$ for $n \geq 4$ qubits.

2.2 Simulation Methods

2.2.1 State Vector

Type: Exact. Represents a pure quantum state as a complex amplitude vector:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \quad \sum_i |\alpha_i|^2 = 1 \tag{2}$$

Memory scales as $O(2^n)$, storing 2^n complex numbers (16 bytes each for double precision). This is the most memory-efficient exact simulation method. **Choose when:** Developing and debugging quantum algorithms on pure states without noise—the default choice for algorithm research up to ∼30 qubits.

2.2.2 Density Matrix

Type: Exact. Represents mixed quantum states as:

$$\rho = \sum_k p_k |\psi_k\rangle \langle\psi_k| \tag{3}$$

Memory scales as $O(4^n)$, storing a $2^n \times 2^n$ complex matrix. **Choose when:** Modeling noise, decoherence, or open quantum systems where the state cannot be described by a pure state vector. Essential for realistic hardware simulation but limited to ∼14 qubits.

2.2.3 Matrix Product State (Tensor Network)

Type: Approximate (tunable accuracy via bond dimension χ). Represents the state as a chain of tensors:

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n} |i_1 \cdots i_n\rangle \tag{4}$$

Memory scales as $O(n \cdot \chi^2 \cdot d)$ where $d = 2$ for qubits—*linear* in the number of qubits. States with high entanglement require large χ for accuracy, potentially losing the memory advantage. **Choose when:** Simulating many qubits (> 30) with limited entanglement, such as 1D quantum systems, shallow circuits, or variational algorithms where entanglement grows slowly.

2.2.4 Unitary Matrix

Type: Exact. Computes the full transformation matrix U such that $|\psi'\rangle = U|\psi\rangle$. Memory scales as $O(4^n)$, practical up to ∼12 qubits. **Choose when:** Verifying gate implementations, debugging circuit constructions, or when the same circuit will be applied to many different input states (compute U once, reuse for all inputs).

2.2.5 Pulse-Level (Hamiltonian) Simulation

Type: Exact (numerically, within solver tolerance). Simulates quantum dynamics through time evolution of the system Hamiltonian rather than abstract gate operations. The state evolves according to the Schrödinger equation:

$$i\hbar\frac{d|\psi\rangle}{dt} = H(t)|\psi\rangle \tag{5}$$

where $H(t)$ includes control pulses that implement gate operations. Memory scales as $O(4^n)$ for the full density matrix evolution. **Choose when:** Designing optimal control pulses, analyzing hardware timing constraints, studying the effects of pulse imperfections, or bridging the gap between abstract circuits and physical implementations.

2.3 Literature Review

IBM's Qiskit framework provides multiple simulation backends through Qiskit Aer [1], including the matrix product state method for tensor network simulation. Grover's search algorithm [2] provides the foundation for many quantum database search applications. Tensor network methods for quantum simulation have been extensively studied [3], offering advantages for circuits with limited entanglement. QuTiP [4] provides Hamiltonian-level simulation capabilities through its qutip-qip extension, enabling pulse-level quantum circuit simulation.

3 Methodology

3.1 Tools and Technologies

- **Python 3.13**: Programming language
- **Qiskit 1.3**: Quantum circuit construction
- **Qiskit Aer**: State-vector, density matrix, MPS, and unitary simulators
- **QuTiP / qutip-qip**: Pulse-level Hamiltonian simulation
- **NumPy**: Numerical operations

3.2 Approach

The approach involves:

1. Constructing an n-qubit Grover's search circuit with a configurable target state
2. Building the oracle to mark the target state $|t\rangle$
3. Using `qiskit.circuit.library.GroverOperator` for the oracle+diffuser combination
4. Running the circuit through five simulation methods
5. Verifying that the target state has the highest measurement probability
6. Measuring execution time and memory usage

3.3 Memory Safety

To prevent crashes on consumer hardware:

- Density matrix simulation is skipped for $n > 14$ qubits
- Unitary simulation is skipped for $n > 12$ qubits
- Pulse-level simulation is skipped for $n > 2$ qubits (multi-controlled gate complexity)

4 Implementation

4.1 Grover's Circuit

Grover's algorithm structure for searching target state $|t\rangle$:

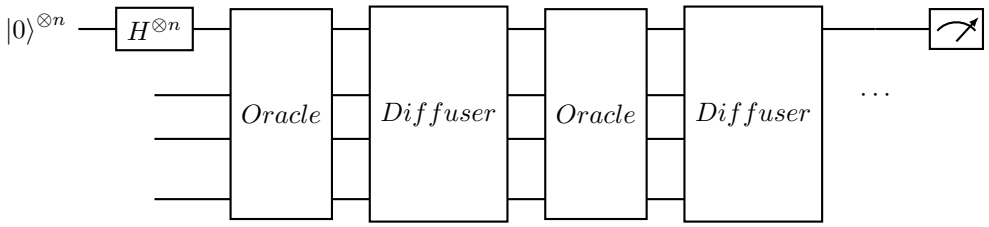

Figure 1: Grover's algorithm. Oracle marks target, Diffuser amplifies. Repeat $\lfloor \pi\sqrt{N}/4 \rfloor$ times.

4.2 System Design

```
grover_comparison.py
  |-- build_oracle(n_qubits, target)
  |-- build_grover_circuit(n_qubits, target, iterations)
  |-- theoretical_success_prob(n_qubits, iterations)
  |-- run_statevector(circuit)
  |-- run_density_matrix(circuit, n_qubits)
  |-- run_mps(circuit)
  |-- run_unitary(circuit, n_qubits)
  |-- run_qutip_pulse(n_qubits, target, iterations)
  |-- verify_results(results, n_qubits, target)
  |-- compare_results(results, n_qubits)
  |-- main()  # with argparse --qubits --target --method
```

4.3 Oracle Construction

The oracle must flip the phase of the target state $|t\rangle$ while leaving all other states unchanged: $|t\rangle \to -|t\rangle$. The key insight is that a multi-controlled Z (MCZ) gate applies a phase flip only when *all* control qubits are in state $|1\rangle$. To mark an arbitrary target state:

1. **Transform target to all-ones:** Apply X gates to qubits where the target's binary representation has a 0. This maps $|t\rangle \to |11\ldots1\rangle$ while permuting other basis states.

2. **Apply MCZ:** The multi-controlled Z gate flips the phase of $|11\ldots1\rangle$ only.

3. **Undo transformation:** Apply the same X gates again to restore the original basis, now with $|t\rangle$ having a flipped phase.

Since MCZ is not a native gate, we decompose it using the identity $\text{CZ} = (I \otimes H) \cdot \text{CNOT} \cdot (I \otimes H)$, which generalizes to $\text{MCZ} = (I^{\otimes n-1} \otimes H) \cdot \text{MCX} \cdot (I^{\otimes n-1} \otimes H)$.

```python
def build_oracle(n_qubits, target):
    oracle = QuantumCircuit(n_qubits)
    target_binary = format(target, f'0{n_qubits}b')
    # Apply X gates where target bit is 0
    for i, bit in enumerate(target_binary[::-1]):
        if bit == '0':
            oracle.x(i)
    # Multi-controlled Z gate via H-MCX-H
    oracle.h(n_qubits - 1)
    oracle.mcx(list(range(n_qubits - 1)), n_qubits - 1)
    oracle.h(n_qubits - 1)
    # Undo X gates
    for i, bit in enumerate(target_binary[::-1]):
        if bit == '0':
            oracle.x(i)
    return oracle
```

Listing 1: Grover Oracle for Target State

4.4 Pulse-Level Simulation

```python
def run_qutip_pulse(n_qubits, target, num_iterations):
    from qutip_qip.circuit import QubitCircuit
    from qutip_qip.device import LinearSpinChain
    from qutip import basis, tensor

    qc = QubitCircuit(n_qubits)
    # Build circuit with SNOT (Hadamard), X, CNOT gates
    for i in range(n_qubits):
        qc.add_gate("SNOT", targets=i)
    # ... Grover iterations with CZ decomposition

    processor = LinearSpinChain(n_qubits)
    processor.load_circuit(qc)
    init_state = tensor([basis(2, 0) for _ in range(n_qubits)])
    result = processor.run_state(init_state=init_state)
    return result.states[-1].full().flatten()
```

Listing 2: QuTiP Pulse-Level Simulation

5 Results and Analysis

5.1 Verification Results

Grover's search was tested with target state $|00101010\rangle$ (decimal 42) on 8 qubits:

Table 1: Grover's Search Verification (8 qubits, target = 42, 13 iterations)

Method	Target Prob	Max State	Max Prob	Status
State-vector	0.9862	$\|00101010\rangle$	0.9862	PASS
Density Matrix	0.9862	$\|00101010\rangle$	0.9862	PASS
MPS (Tensor Net)	0.9862	$\|00101010\rangle$	0.9862	PASS
Unitary	0.9862	$\|00101010\rangle$	0.9862	PASS

5.2 Output Probability Distribution

Grover's search amplifies the target state:

```
Top 4 states by probability:
  1. |00101010> (target): prob=0.986186
  2. |10101010>         : prob=0.000054
  3. |01011010>         : prob=0.000054
  4. |00110001>         : prob=0.000054
```

5.3 Pulse-Level Simulation (2 Qubits)

For pulse-level simulation via QuTiP, we use a 2-qubit Grover circuit (target $|11\rangle$, 1 iteration):

Table 2: Pulse-Level vs State-Vector Comparison (2 qubits, target = 3, 1 iteration)

Method	Target Prob	Max State	Max Prob	Status
State-vector	1.0000	$\|11\rangle$	1.0000	PASS
QuTiP Pulse	1.0000	$\|11\rangle$	1.0000	PASS

The pulse-level simulation achieves identical results to the ideal state-vector simulation, demonstrating that the Hamiltonian evolution correctly implements the quantum gates. The pulse simulation uses the LinearSpinChain processor model with SNOT (Hadamard) and CNOT gates decomposed into native pulse sequences.

5.4 Performance Comparison

Table 3: Simulation Method Comparison (8-qubit Grover's, target = 42)

Method	Time (ms)	Memory	Scaling
State-vector	537.6	4.0 KB	$O(2^n)$
Density Matrix	2570.1	1.0 MB	$O(4^n)$
MPS (Tensor Net)	674.8	4.0 KB	$O(n \cdot \chi^2)$
Unitary	66.2	1.0 MB	$O(4^n)$

5.5 Analysis

State-vector simulation showed longer initial timing due to JIT compilation overhead in the Aer backend. For repeated simulations, this overhead amortizes. Memory usage is optimal at $O(2^n)$, making it suitable for up to ∼30 qubits on typical hardware.
Density matrix simulation requires 4× more memory than state-vector per qubit added. Required when modeling noise, decoherence, or mixed states. Becomes impractical beyond 14 qubits.
MPS (Tensor Network) demonstrated similar output memory to state-vector but uses a fundamentally different internal representation. The bond dimension χ controls accuracy vs. memory trade-off. Particularly advantageous for circuits with low entanglement or when simulating many qubits approximately.
Unitary simulation computes the full transformation matrix, useful for verifying gate implementations. Has the same memory constraints as density matrix. Despite both unitary and density matrix having $O(4^n)$ memory scaling, unitary simulation is significantly faster (66ms vs 2570ms) because:

- **Operation complexity:** Unitary simulation applies gates via matrix multiplication $U' = G \cdot U$, which is $O(4^n)$ per gate. Density matrix evolution requires $\rho' = G\rho G^\dagger$, involving *two* matrix multiplications per gate—effectively 2× the operations.
- **Final state extraction:** Unitary simulation computes probabilities by a single matrix-vector product $|\psi'\rangle = U|0\rangle$ followed by $|\psi_i'|^2$. Density matrix must extract the diagonal of the full $2^n \times 2^n$ matrix.
- **Backend optimization:** Qiskit Aer's unitary simulator is optimized for computing the circuit's transformation matrix without intermediate state tracking, while density matrix simulation must maintain the full density operator throughout.

Table 4: Simulation Methods: Use Cases

Method	Best Use Cases	Limitations
State-vector	Algorithm development, pure state simulation, $n < 30$	No noise modeling
Density Matrix	Noise characterization, decoherence, open systems	$n < 15$ due to memory
MPS	Large qubit counts, low-entanglement circuits	Accuracy depends on χ
Unitary	Gate verification, circuit debugging	Same limits as density matrix
Pulse-Level	Hardware timing analysis, control pulse design	$n < 3$ in this implementation

6 Challenges Encountered

- **Circuit Depth:** Grover's algorithm requires multiple oracle+diffuser iterations, creating deeper circuits than QFT. This increases gate count and complexity.
- **Memory Scaling:** Density matrix and unitary methods become infeasible beyond 14 qubits. Implemented automatic skip guards to prevent crashes.
- **Method Selection:** Different simulation methods have vastly different trade-offs. Added `-method` CLI flag to allow selective execution.
- **Pulse-Level Gate Decomposition:** The CZ gate required for Grover's oracle is not natively available in QuTiP's LinearSpinChain processor. Decomposed CZ into H-CNOT-H sequence to work within the processor's native gate set.

7 Conclusion

This comparison of five quantum circuit simulation methods on Grover's search algorithm reveals distinct trade-offs:

1. **State-vector** simulation is optimal for pure-state algorithm development with memory scaling $O(2^n)$. Achieved 98.6% success probability matching theory.
2. **Density matrix** simulation enables noise modeling but scales as $O(4^n)$, limiting practical use to ∼14 qubits. At 8 qubits, requires 1 MB and 2.5 seconds.
3. **MPS (Tensor Network)** offers memory efficiency matching state-vector while using tensor decomposition internally.
4. **Unitary** simulation is specialized for gate verification, with fastest execution but high memory cost.
5. **Pulse-level** simulation via QuTiP provides Hamiltonian-based evolution, giving insight into hardware-level dynamics. Limited to 2 qubits in this implementation due to multi-controlled gate decomposition complexity.

Grover's algorithm works well for comparing simulators because different targets produce visibly different results, and the clear success criterion (target state probability) makes verification straightforward. All methods achieved the expected success probability, validating theoretical predictions.

References

[1] Qiskit contributors, "Qiskit: An Open-source Framework for Quantum Computing," 2025. [Online]. Available: https://qiskit.org/

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annual ACM Symposium on Theory of Computing*, 1996, pp. 212–219.

[3] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics*, vol. 349, pp. 117–158, 2014.

[4] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems," *Computer Physics Communications*, vol. 183, no. 8, pp. 1760–1772, 2012.