A Performance Comparison of Statevector and MPS-based Quantum Emulators

Daniel Maroto Sánchez

Delft University of Technology

TU Delft

A Performance Comparison of Statevector and MPS-based Quantum Emulators

by

Daniel Maroto Sánchez

Supervisors: M. Möller and A. Bravo Montes
Project Duration: 04, 2025 - 1, 2026
Faculty: Faculty of Electrical Engineering, Mathematics & Computer Science, Delft

TU Delft

Preface

This thesis is dedicated to my father, whose guidance and inspiration have shaped my path. I am certain he would be incredibly proud of this accomplishment.

Daniel Maroto Sánchez
Delft, January 2026

Summary

This thesis explores the Quantum Approximate Optimization Algorithm (QAOA) and its recursive variant (RQAOA), focusing on their performance on different graph topologies and the challenges of simulating these algorithms. Key concepts of quantum computing are introduced to provide the necessary background. The work examines methods for emulating quantum circuits, including statevector and Matrix Product State (MPS) approaches, and considers the impact of noise on algorithmic behavior. It also investigates how circuit size, depth, and graph structure affect algorithm performance and emulator efficiency, including tests on pseudo-random circuits comparing statevector and MPS emulations. Overall, the thesis provides a comprehensive study of QAOA, RQAOA, and quantum circuit simulation techniques, highlighting both methodological considerations and directions for future research in quantum optimization.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
A2A	All-to-All
CCNOT	Controlled-CNOT
CNOT	Controlled-NOT
CPU	Central Processing Unit
CSV	Comma-separated Value
CZ	Controlled-Z
DMRG	Density Matrix Renormalization Group
EoF	Entanglement of Formation
E_C	Entanglement Cost
E_D	Distillable Entanglement
E_N	Logarithmic Negativity
E_R	Relative Entropy of Entanglement
GHZ	Greenberger-Horne-Zeilinger
GPU	Graphic Processing Unit
HPC	High Performance Computing
LIMDD	Local Invertible Map Decision Diagrams
LOCC	Local Operations and Classical Communication
MPS	Matrix Product State
NISQ	Noisy Intermediate-Scale Quantum
PEPS	Projected Entangled Pair States
PIMC	Path Integral Monte Carlo
PQC	Parameterized Quantum Circuit
QASM	Quantum Assembly Language
QAOA	Quantum Approximate Optimization Algorithm
QMDD	Quantum Multiple-valued Decision Diagrams
QPU	Quantum Processing Unit
RQAOA	Recursive QAOA
SQA	Simulated Quantum Annealing
SPAM	State Preparation and Measurement
SSH	Secure Shell
TB	TeraByte
TEBD	Time-Evolving Block Decimation
TNs	Tensor-Networks
TTNs	Tree Tensor Networks
VQA	Variational Quantum Algorithm
VQE	Variational Quantum Eigensolver
VPN	Virtual Private Network
SVD	Singular Value Decomposition

1

Introduction

Since the concept of quantum computing was introduced in the 1980s by Benioff and Feynman [1], the field has advanced considerably, evolving from a conceptual and theoretical framework into a rapidly developing area with tangible prospects and real-world applications. It is well established that classical computers process information using binary bits, represented as 0s and 1s. In contrast, quantum computing introduces a new paradigm, now commonly used, known as the qubit. This qubit will represent our data as 0s, 1s and a *superposition* of both states [2]. Moreover, multiple qubits can exhibit a phenomenon known as *entanglement*. In this situations, the qubits become interdependent, such that the state of one cannot be described independently of the others, regardless of the distance separating them. Notably, these types of correlations cannot be fully characterize by classical information theory [3]. However, the practical use of the previous properties is limited by a phenomenon called *decoherence*. Decoherence arises from the interaction of qubits with their environment, leading to the loss of quantum coherence and making the system behave as a classical system [2].

The potential of quantum computing stems from its ability to exploit the principles of quantum mechanics, enabling computational speedups over classical methods for specific problems. Well-known examples include Shor's factoring algorithm and Grover's unstructured search algorithm [1], offering exponential and quadratic speedups over the best classical algorithms for cryptography and database search. Over the past years, developments in quantum computing have paved the way for practical applications in various domains. Quantum techniques are being explored in areas such as chemistry, materials science, optimization, machine learning, and secure communication [4], indicating that the applications of quantum computing are not limited to theoretical algorithms.

In fact, industries are increasingly investing into quantum computing, eager not to be left behind—a situation reminiscent of the early 2000s race for graphics processing units (GPUs). Looking ahead, McKinsey projects that the quantum computing market could generate between $28 billion and $72 billion in revenue by 2035, with the total quantum technology market potentially reaching $97 billion by the same year [5]. These developments highlight growing commercial interest and the push to translate quantum computing from theory into practical applications.

Despite this investment, current quantum devices remain expensive, noisy and limited in scale [6]. They require extensive error correction and specialized infrastructure to improve performance, and are generally not well suited for testing, algorithm development or educational purposes, which has fueled interest in classical quantum emulators. Leading technology providers—IBM with Qiskit Aer [7], Google with Cirq [8], and startups such as Quantinuum [9] and Xanadu's PennyLane [10]—have developed statevector [11] and Tensor-Network (TN) emulators [12, 13] to support algorithm development and verification.

Building on this foundation, the earliest quantum circuit emulators were based on statevector methods, which explicitly represent the evolution of the full 2^n-dimensional wavefunction of an n-qubit system throughout a quantum circuit. Although statevector emulation has the advantage of being exact, it comes at the cost of significant computational resources, both in memory—requiring storage of 2^n

complex amplitudes for an n-qubit state—and in runtime, which scales exponentially with the number of qubits [14]. This makes it impossible to run circuits of interest that could truly demonstrate a quantum advantage on conventional computers; therefore, the integration of quantum processing units (QPUs) and emulators into high-performance computing (HPC) environments, such as Cuda-Q from NVIDIA [15], QuEST [16] or the Qaptiva appliance from Eviden [17], is becoming increasingly common [18].

To overcome the limitations of statevector emulators, significant development has been made in various types of classical quantum emulators. These include density matrix-based emulators, TN-based methods, stabilizer or Clifford emulators, and approaches based on binary decision diagrams [19, 20, 21, 22, 23] .

Among these, TN techniques—particularly matrix product states (MPS)—have become one of the most widely used approaches for emulating quantum circuits with reduced computational resources. MPS methods where first introduced by Vidal in his time-evolving block decimation (TEBD) algorithm [24]. MPS techniques exploit the fact that many quantum systems—particularly one-dimensional circuits with limited entanglement—can be efficiently represented as tensor networks, reducing the memory and computational cost from exponential to polynomial scaling under controlled truncation of bond dimensions. The accuracy of the emulation depends on the bond dimension and truncation scheme, which restricts the expressibility of highly entangled states [25].

Each of the aforementioned emulation techniques has distinct advantages and limitations, which implies that no single method is universally applicable to all classes of quantum circuits or paradigms of quantum computing. As a result, selecting an appropriate emulator requires careful consideration of the specific context in which it will be applied. This decision is influenced by several parameters, such as circuit size and depth, the number of qubits involved, the target application, underlying computational paradigm, etc. The interrelation among these factors underscores the importance of evaluating the capabilities and suitability of existing emulators to identify their performance in different domains of applicability.

Among the various quantum emulation approaches discussed, this work will focus specifically on statevector and TN emulators, with the latter represented by the MPS method. This choice is motivated by the fact that the emulations will be carried out using the Qaptiva Appliance provided by Eviden [17], which offers efficient implementations of these two models. In addition, these emulators do not require the tuning of a large number of hyperparameters, unlike the Simulated Quantum Annealing (SQA) emulator, which would make the comparison harder due to its strong dependence on parameter adjustment. Furthermore, limiting the scope to a two-party comparison allows for a clearer and more focused analysis, avoiding unnecessary complexity that would arise from including additional emulator types.

In this thesis, we investigate emulator performance by benchmarking both statevector and MPS emulators on different circuit families. For statevector methods, we focus on the Quantum Approximate Optimization Algorithm (QAOA) [26] and its recursive variant (RQAOA) [27], while for MPS emulation we employ a family of randomly generated circuits, used in [28], to study low-entanglement circuits and how entanglement evolve in those circuits. Finally, recognizing the impact of realistic noise, we will incorporate appropriate noise models into the statevector emulations to better approximate the behavior of near-term quantum hardware, following the guidelines from [29].

Based on the considerations above, this work aims to address several research questions related to the performance and behavior of quantum emulators when executing variational algorithms. First, we investigate how the QAOA compares with RQAOA across different types of graph structures, extending the analysis beyond the studied complete graphs using the statevector emulator. Given that QAOA tends to generate significant entanglement, which increases with the circuit depth, we also aim to examine whether the statevector emulator indeed performs better than the MPS emulator under such conditions, and whether there exists a regime where the MPS approach can surpass the statevector one. Furthermore, we explore to what extent the MPS emulator can handle a higher number of qubits, and for a fixed system size, what circuit depth marks the point at which MPS becomes more efficient than statevector. Finally, we assess the effect of introducing realistic noise into the statevector emulations, examining whether such noise degrades the performance of QAOA and RQAOA or if these algorithms exhibit robustness against it.

Objectives

Based on the considerations discussed above, this work is guided by the following research objectives:

- To determine how the performance of the QAOA compares with that of its recursive variant, RQAOA, across different graph topologies, and under same optimization conditions, extending beyond the already studied case of complete graphs.
- To assess the effect of incorporating realistic noise into statevector emulations and to evaluate whether such noise significantly impacts the performance of QAOA or if this algorithm exhibit robustness against it.
- To investigate whether, due to the higher levels of complexity generated by QAOA at increasing circuit depths, the statevector emulator consistently outperforms the MPS emulator, or whether a regime exists in which the MPS approach achieves superior efficiency.
- To examine how the number of qubits and circuit depth influence the relative scalability and computational efficiency of both emulation methods, identifying the crossover point where MPS becomes more advantageous than statevector.

This thesis is organized as follows. Chapter 2 provides the necessary background on quantum computing, including the fundamentals of qubits, quantum gates, circuits, algorithms, and emulation techniques, as well as a discussion of noisy emulations and related work. Chapter 3 presents the development environment, detailing the selected emulators and their functionalities. Chapter 4 describes the proposed solution, including the methodology, implementation, and analysis workflow. Chapter 5 presents the results of the experiments, comparing different algorithms, emulators, and graph topologies. Finally, Chapter 6 summarizes the main conclusions and outlines potential directions for future work.

2

Background

This chapter establishes the theoretical foundation necessary to address the research questions outlined in the introduction. It presents an overview of the key principles underlying quantum computing, quantum circuits, entanglement, quantum emulators, and noise, followed by a discussion of related works in the field. By reviewing the relevant literature on these topics, the chapter situates the present study within the broader context of quantum computation and emulation. In doing so, it emphasizes the mechanisms that influence emulator performance and clarifies how these aspects relate directly to the comparative analysis conducted in this thesis.

2.1. Basics of Quantum Computing

2.1.1. Qubits

The quantum-bit or **qubit** is the fundamental unit of information in quantum computing. Unlike a classical bit, which can exist only in one of two definite states—0 or 1—a qubit can exist in a *superposition* of both states simultaneously, as depicted in Figure 2.1. Mathematically, it is described as a normalized vector in a 2-dimensional complex Hilbert space, expressed as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{2.1}$$

where α and β are called amplitudes, satisfying $|\alpha|^2 + |\beta|^2 = 1$. For a single qubit, the 2-dimensional Hilbert space has two independent basis states. The computational basis vectors $|0\rangle$ and $|1\rangle$ are simply labels for these basis vectors, which can be represented explicitly as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

According to Born's rule [30], the probability of obtaining a specific measurement outcome is given by the squared modulus of its corresponding amplitude. Therefore, when the qubit is measured in the computational basis, the outcomes $|0\rangle$ and $|1\rangle$ occur with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively.

The representation of a qubit as a linear combination of basis states gives rise to several uniquely quantum phenomena. First, **superposition** allows a qubit to exist in multiple states simultaneously, which enables quantum computers to explore many computational paths at once. Real quantum systems, however, are subject to **decoherence**, which tends to destroy superpositions and limit the fidelity of computations. Finally, qubits can become entangled. **Entanglement** is a central resource for many quantum algorithms and will be discussed in detail in Section 2.3.

2.1.2. Quantum Gates

Just as classical bits can be manipulated by logical gates (e.g., AND, OR, NOT), qubits are manipulated by **quantum gates**. Quantum gates are the building blocks of quantum circuits, performing specific

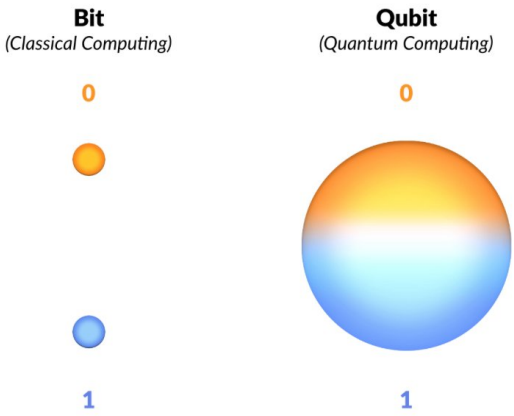


Figure 2.1: Classical Bit vs Quantum Bit. Source: Quantum Computing at Davis, The Qubit [31].

unitary operations on one or more qubits. Mathematically, a quantum gate U is a square matrix satisfying:

$$U^\dagger U = UU^\dagger = I, \tag{2.2}$$

where $U^\dagger$ is the conjugate transpose of U and I is the identity matrix, this is the definition of an **unitary matrix**. Unitarity ensures that the evolution of a quantum system is reversible and preserves the total probability, i.e., the norm of quantum states remains 1. Consequently, quantum gates must be reversible, allowing the input state to be reconstructed from the output in Figure 2.2, so that no information is lost during computation [14].

(a) Single-qubit unitary transformation

(b) Reversible unitary operation

Figure 2.2: Comparison of a standard unitary transformation (left) with its corresponding reversible operation (right).

Quantum gates can be classified by the number of qubits they act upon, i.e., their arity. Some common quantum gates and their matrix representations are:

- **Single-qubit gates:**
  - *Pauli gates:*

    $$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ (Bit-flip)}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \text{ (Bit and phase flip)}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ (Phase-flip)}$$

    where i is the imaginary unit.
  - *Hadamard gate:*

    $$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ (Creates superposition)}$$

  - *Rotation gates:*

    $$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix},$$
    $$R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \quad \text{(Rotations of angle } \theta \text{ around X, Y and Z axes, respectively)}$$
    $$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

- **Two-qubit gates:**
  - *Controlled-NOT (CNOT) and SWAP gates:*

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Three-qubit gates:**
  - *Toffoli or Controlled-CNOT (CCNOT) gate:*

$$\text{Toffoli} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{(Flips target if both controls are 1)}$$

The circuit representation of some of the gates mentioned above is shown in Figure 2.3.

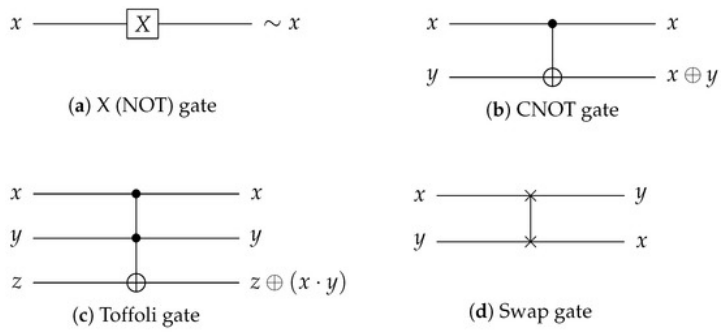


Figure 2.3: Quantum circuit representation of selected single- and multi-qubit gates. Image taken from [32]

As can be seen, the size of the matrices grows exponentially with the number of qubits involved. In an n-qubit system, a quantum gate is represented by a $2^n \times 2^n$ matrix. This exponential scaling will be particularly important later when discussing the performance and limitations of quantum emulators.

By combining gates in sequence, complex quantum algorithms can be implemented, allowing the manipulation of quantum states for computational tasks beyond the reach of classical circuits.

2.1.3. Quantum Circuits: Fundamentals

A **quantum circuit** is a model for quantum computation in which qubits are initialized, transformed by quantum gates, and finally measured. The typical structure of a quantum circuit consists of three main parts:

1. **Initialization:** All qubits are usually initialized in the computational basis, commonly in the state $|0\rangle^{\otimes n}$. For example, a two-qubit system would start in $|00\rangle$.
2. **Quantum operations:** The initialized qubits are transformed through a sequence of quantum gates. Each gate modifies the state of the system, producing intermediate states which we can denote as $|\phi_1\rangle$, $|\phi_2\rangle$, etc.
3. **Measurement:** At the end of the circuit, qubits are measured in a chosen basis (commonly the computational basis). The measurement collapses the quantum state to classical outcomes, with probabilities determined by the amplitudes of the final state.

The **circuit depth** of a quantum circuit is defined as the maximum number of successive gates applied to any individual qubit, and it serves as an important measure of computational complexity in quantum computing. Furthermore, when considering only the unitary portion of a quantum circuit—excluding measurements, resets, and other non-unitary operations—the sequence of quantum gates composes into a single unitary operator U acting on the entire qubit register. In this sense, a quantum circuit can be viewed as a large unitary transformation that incorporates all of its gate operations.

A simple two-qubit circuit illustrating these steps is shown in Figure 2.4. The states after each operation are labeled to show the evolution of the system.

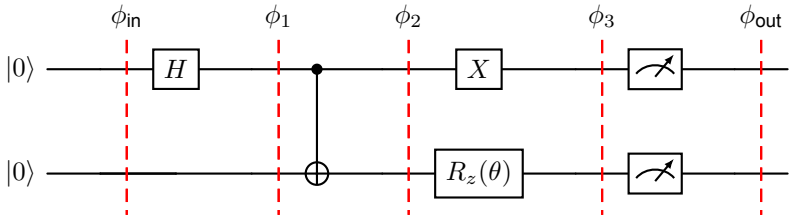


Figure 2.4: Example of a two-qubit quantum circuit. The evolution of the quantum state is tracked by the initial state ϕ_{in} and intermediate states ϕ_1, ϕ_2, ϕ_3, labeled above the dashed lines, with ϕ_{out} being the final (classical) state after measurement.

2.1.4. Measurements in Quantum Circuits

Measurement is the process by which quantum information is converted into classical information. Upon measurement, the quantum state of a qubit (or set of qubits) *collapses* to one of the eigenstates of the measurement operator, with probabilities given by the squared modulus of the amplitudes (Born's rule).

It is important to note that measurements are inherently non-unitary: unlike quantum gates, they are irreversible and destroy superposition in the measured qubits, turning them into classical bits.

In the context of quantum circuits, there are generally two types of measurement outcomes:

- **Projective (bitstring) measurements:** The simplest type of measurement, usually performed in the computational basis. Each qubit collapses to either $|0\rangle$ or $|1\rangle$, and the combined outcome of n qubits is a classical *bitstring*. By sampling many times, one can estimate probabilities of different outcomes and reconstruct statistics of the quantum state.
- **Expectation value measurements:** Instead of obtaining individual bitstrings, one can measure the *average value* of an observable over the quantum state. An *observable* is represented by a Hermitian operator, i.e., a matrix H satisfying

$$H = H^\dagger. \tag{2.3}$$

  Hermitian operators have real eigenvalues, which correspond to the possible outcomes of a measurement. A Hamiltonian is a special Hermitian operator that encodes the energy of a quantum system. Measuring the expectation value of a Hamiltonian, $\langle\psi|H|\psi\rangle$, allows one to estimate the energy of the system or optimize a cost function in variational algorithms (see Section 2.2.1). This type of measurement is central in quantum chemistry, physics simulations, and variational quantum algorithms [26].

2.1.5. Quantum Algorithms

Quantum algorithms are sequences of quantum gates and measurements designed to solve computational problems by exploiting the principles of quantum mechanics.

Many quantum algorithms, particularly in the variational and optimization context, are associated with a cost function that quantifies the quality of a given quantum state with respect to the problem of interest. The cost function is often defined as the expectation value of an operator (e.g., a Hamiltonian H):

$$C(\boldsymbol{\psi}) = \langle\psi|H|\psi\rangle \tag{2.4}$$

where ψ is the quantum state.

The goal is to minimize or maximize the cost function, i.e. finding the ground state of H.

In other algorithms, such as quantum search or combinatorial optimization, the cost function may correspond to the probability of measuring a desired solution (bitstring) in the computational basis. By designing circuits that amplify the probability of favorable outcomes, quantum algorithms can achieve advantages over classical approaches.

2.2. Quantum Circuits for Algorithms

Some quantum algorithms are implemented through dedicated circuits specifically designed to solve particular problems [33], while others utilize more general circuit architectures to explore quantum states or facilitate algorithmic flexibility. In addition, certain circuit types serve specialized purposes, such as variational circuits for optimization tasks or randomized constructions for benchmarking and research. This section introduces some of these circuit types, emphasizing their design principles and typical applications in current research tasks.

2.2.1. Variational Quantum Algorithms

Variational Quantum Algorithms (VQAs) are a class of *hybrid quantum-classical algorithms* designed to exploit near-term quantum devices, which presents significant limitations in terms of qubit counts, gate fidelities, and coherence times [34]. The core idea behind a VQA is to iteratively optimize a *parameterized quantum circuit* (PQC) using a classical optimizer to minimize or maximize a cost function, representing our problem and thus, exploiting the variational principle [35]. A VQA consists of three main components:

1. **Parameterized Quantum Circuit:** The PQC $U(\vec{\theta})$ is a parameterized quantum circuit that generates quantum states of the form

$$|\psi(\vec{\theta})\rangle = U(\vec{\theta})|0\rangle^{\otimes n}, \tag{2.5}$$

   where $\vec{\theta} = (\theta_1, \ldots, \theta_m)$ denotes the set of tunable parameters. The circuit is typically structured in layers, each composed of gates whose actions depend on the parameters $\vec{\theta}$; this layered architecture is referred to as the *ansatz*. The choice of ansatz can be either general-purpose — such as *hardware-efficient ansatz* [36], which are specifically designed to minimize circuit depth by adapting to the native gate set of the hardware — or problem-specific, as in the case of the QAOA ansatz [26].

2. **Cost Function:** The cost function $C(\vec{\theta})$ quantifies how well the parameterized state solves the problem. It is usually defined as the expectation value of a Hermitian operator H:

$$C(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle. \tag{2.6}$$

   This operator can represent, for example, a Hamiltonian in quantum chemistry or an objective function in combinatorial optimization.

3. **Classical Optimizer:** The parameters $\vec{\theta}$ are iteratively updated using a classical optimization routine to minimize or maximize the cost function. Common choices include gradient-free methods such as *Nelder-Mead* [37] and *COBYLA* [38], as well as gradient-based methods like *Adam* [39].

This hybrid framework allows VQAs to address a variety of problems, including *quantum chemistry, combinatorial optimization, and quantum machine learning* [40], while keeping the quantum circuit depth manageable for near-term quantum devices. The flexibility of VQAs arises from the choice of PQC, cost function, and optimization strategy, which together determine the performance and suitability for a given application.

Barren Plateaus

Barren plateaus are regions in the optimization landscape of VQAs where the gradients of the cost function with respect to the circuit parameters vanish exponentially with the number of qubits or circuit

depth [41]. This makes local optimization extremely difficult, as methods like gradient-based or gradient-free cannot reliably identify informative search directions because the the cost function becomes nearly flat over large portions of the parameter space.

Formally, for a cost function $C(\boldsymbol{\theta})$ depending on variational parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$, the variance of the gradient typically scales as

$$\text{Var}\left[\frac{\partial C}{\partial \theta_i}\right] \sim \mathcal{O}\left(\frac{1}{2^n}\right), \tag{2.7}$$

where n is the number of qubits and $\mathcal{O}$ is the asymptotic growth rate of the function. This exponential decay implies that, for large systems, the expected gradient approaches zero, creating an almost flat landscape.

In this context, it is important to note that overly simple circuits may fail to represent useful quantum states, whereas highly expressive circuits can introduce optimization challenges. Sim et al. [42] provide a detailed study on how different PQCs vary in expressibility and entangling capacity, offering an explanation for this trade-off.

An example of a VQA is the Variational Quantum Eigensolver (VQE). In VQE, the ansatz is flexible and problem-dependent, with common choices including hardware-efficient circuits or the unitary coupled cluster ansatz used in quantum chemistry [43]. Another example is the QAOA, which can be seen as a specialized form of VQE, where the ansatz is structured according to the problem Hamiltonian of a combinatorial optimization task. We will now focus on the QAOA and its variation, the RQAOA, which will be used in the comparisons.

Quantum Approximate Optimization Algorithm

QAOA, proposed by Farhi et al. [26], is one of the most studied variational quantum algorithms. This hybrid quantum-classical approach is mainly designed to solve combinatorial optimization problems. The QAOA ansatz has a fixed structure, it is constructed by alternating between two types of unitaries. The first is the problem unitary, which encodes the optimization problem via the cost Hamiltonian H_C:

$$U_C(\gamma) = e^{-i\gamma H_C}, \tag{2.8}$$

and the second is the mixer unitary, which induce the exploration of the solution space through the mixing Hamiltonian H_M:

$$U_M(\beta) = e^{-i\beta H_M}. \tag{2.9}$$

where H_M is typically chosen as a sum of Pauli-X operators. In the QAOA, we can tune the parameter p, which determines the number of alternating layers in the ansatz, controlling the expressiveness of the state: larger p allows the algorithm to better approximate the optimal solution, at the cost of increased circuit complexity. However, there are some scenarios where increasing p does not lead to better solutions, as pointed in [44]. For p layers, the QAOA ansatz prepares a state

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = U_M(\beta_p)U_C(\gamma_p)\cdots U_M(\beta_1)U_C(\gamma_1)|+\rangle^{\otimes n}, \tag{2.10}$$

where $|+\rangle^{\otimes n}$ is the uniform superposition over all computational basis states, obtained by applying $H^{\otimes n}|0\rangle^{\otimes n}$, and $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ are parameters optimized classically to minimize the expectation value of the cost Hamiltonian. We show a representation of the QAOA workflow in Figure 2.5, adapted from [45]. First, the quantum circuit is constructed based on both the problem Hamiltonian and the mixer Hamiltonian, as well as the desired number of layers. Next, the cost function is evaluated, and the optimization is checked for convergence. If the objective has not been met, the parameters are updated and the process is iterated until convergence is achieved.

Recursive QAOA

In this work, we will also explore the Recursive QAOA, introduced by Bae et al. [27]. RQAOA builds upon the standard QAOA by iteratively reducing the size of the problem graph. At each recursion step, a level-p QAOA is applied to the current graph to estimate the most likely edges in the optimal solution. These edges are then fixed by constraints, and the corresponding vertices are merged or removed,

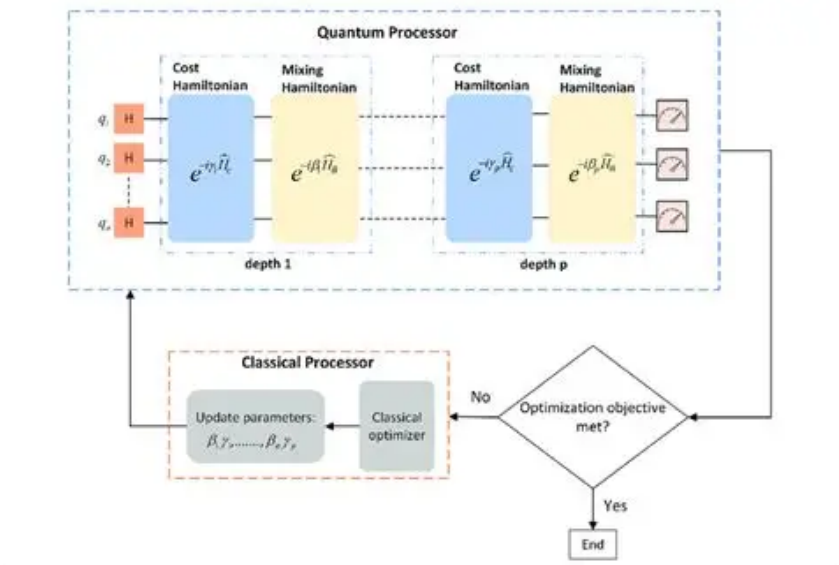


Figure 2.5: Diagrammatic representation of the QAOA workflow.

producing a smaller subgraph corresponding to a smaller cost Hamiltonian. The process is repeated recursively until the graph becomes small enough to be solved exactly, either classically or with a final QAOA step. By progressively reducing the problem size and focusing QAOA on smaller subgraphs, RQAOA can achieve better approximation ratios than standard QAOA for certain problem instances, such as the MaxCut problem (see Section 2.2.3) on complete graphs [27].

We will now briefly delve into the mathematical formulation of RQAOA.

1. Start with the cost Hamiltonian:
$$H_N = \sum_{(i,j)\in E} J_{i,j} Z_i Z_j \tag{2.11}$$
and apply a level-p QAOA to prepare the state
$$|\psi_p(\vec{\beta}^*, \vec{\gamma}^*)\rangle \tag{2.12}$$
which maximizes the following quantity $\langle\psi_p(\beta,\gamma)|H_n|\psi_p(\beta,\gamma)\rangle$.
2. For every edge $(i,j) \in E$, compute
$$M_{i,j} = \langle\psi_p(\vec{\beta}^*, \vec{\gamma}^*)|Z_i Z_j|\psi_p(\vec{\beta}^*, \vec{\gamma}^*)\rangle. \tag{2.13}$$
Then, choose the pair (k,l) which maximizes the magnitude $|M_{i,j}|$.
3. Impose the constraint
$$Z_k = \text{sgn}(M_{k,l}) Z_l \tag{2.14}$$
and replace it into the Hamiltonian
$$H_n = \sum_{(i,k)\in E} J_{i,k} Z_i Z_k + \sum_{i,j\neq k} J_{i,j} Z_i Z_j \tag{2.15}$$
to obtain the reduced Hamiltonian
$$H'_n = \text{sgn}(M_{k,l}) \left[\sum_{(i,k)\in E} J_{i,k} Z_i Z_l \right] + \sum_{i,j\neq k} J_{i,j} Z_i Z_j. \tag{2.16}$$
4. Repeat from 1 on the reduced Hamiltonian H'_n until the problem is small enough to be solved exactly.

As shown in Figure 2.6, RQAOA iteratively reduces the problem graph by applying QAOA at each step. In this example, we begin with a complete graph K_6 containing 6 nodes. After the first iteration, we impose the constraint $x_5 = -x_6$ and substitute it into the original Hamiltonian H_1 . Following this substitution, all edges marked in red in Figure 2.6 are removed, leaving a complete graph K_4 with 4 nodes. This effectively reduces the problem complexity by eliminating 2 nodes in a single step.

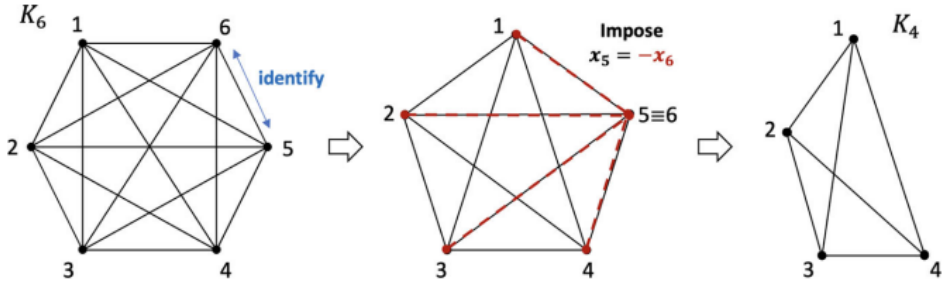


Figure 2.6: Schematic of the RQAOA process, showing recursive graph reduction and QAOA application at each step. Adapted from [27].

2.2.2. Random and Pseudo-Random Circuits

A random or pseudo-random quantum circuit is a quantum circuit designed to approximate the statistical properties of truly random unitaries. To do so, the objective is to sample from the so-called Haar-random unitaries, which are unitary matrices sampled uniformly from the group $U(N)$ according to the Haar measure. While sampling a Haar-random unitary requires exponentially many parameters, pseudo-random circuits emulate their behavior through structured yet randomized layers of local gates [46]. These circuits are particularly useful when benchmarking quantum hardware, studying quantum chaos, and demonstrating computational complexity (e.g., in quantum supremacy experiments) [47].

First, we will briefly introduce what the **Haar measure** is, mainly to provide context for the reader; more details can be found in [48].

The Haar measure is the unique, translation-invariant probability measure on a compact group (such as the unitary group $U(N)$). Translation-invariant means that for any measurable subset $S \subset U(N)$ and any unitary $V \in U(N)$, the measure satisfies

$$\mu_{\text{Haar}}(S) = \mu_{\text{Haar}}(VS) = \mu_{\text{Haar}}(SV),$$

where $VS = \{VU \,|\, U \in S\}$ and $SV = \{UV \,|\, U \in S\}$. This invariance ensures that no particular unitary or basis is preferred, making the Haar measure the natural uniform distribution over $U(N)$.

A **Haar-random unitary** $U \in U(N)$ is then defined as a unitary matrix drawn according to this measure. This definition guarantees that the statistical properties of the sampled unitary are invariant under left or right multiplication by any fixed unitary, a key property of random states or pseudo-random circuits [46].

Once we have defined what Haar-random unitaries are and why they are important, we will make use of this type of circuit in our study. These circuits provide a natural framework to fairly evaluate the performance of our emulators. Specifically, we employ the quantum circuits introduced in [28], which have a fixed structure illustrated in Figure 2.7.

As shown in Figure 2.7, the gray dotted box indicates the structure of a single layer. We need to highlight that in this case D represents the number of layers for this circuit, and it does not correspond to the circuit depth defined previously. Each layer consists of alternating single-qubit gates (colored boxes) and two-qubit gates (dots connected to an empty box). The single-qubit gates are randomly sampled from the set of universal single-qubit gates, generated as

$$e^{i\alpha}\left(\sigma_x \sin\theta\cos\phi + \sigma_y \sin\theta\sin\phi + \sigma_z \cos\theta\right), \quad \alpha, \theta, \phi \in [0, 2\pi),$$

while the two-qubit gates are either CNOTs or controlled-Z (CZs), each applied with equal probability.

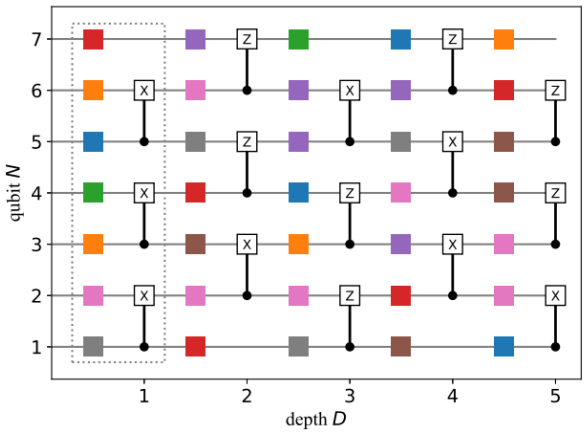


Figure 2.7: Structure of a random quantum circuit layer. Adapted from [28].

For sufficiently large D, these circuits can approximate Haar-random unitaries and generate entanglement efficiently [28].

2.2.3. Combinatorial Problems

Combinatorial problems are a class of mathematical problems in which the goal is to find an optimal configuration or selection from a finite set of possibilities. A typical characteristic of these problems is that the size of the solution space grows rapidly with the number of variables, making exhaustive search infeasible for large instances. Combinatorial problems appear in a wide range of applications, including scheduling, routing, and resource allocation [49]. Notable examples include:

- **Traveling Salesman Problem (TSP):** finding the shortest route that visits each city exactly once and returns to the starting city.
- **Knapsack Problem:** selecting a subset of items with given weights and values to maximize total value without exceeding a weight limit.
- **Graph Coloring:** assigning colors to the vertices of a graph such that no two adjacent vertices share the same color, often with the goal of minimizing the number of colors used.
- **Job Scheduling:** assigning a set of tasks to limited resources over time to optimize a given objective, such as minimizing total completion time or maximizing throughput.
- **MaxCut Problem:** partitioning the vertices of a graph into two sets such that the number of edges between the sets is maximized.

Formally, a combinatorial problem can be described by a tuple (S, C, f), where:

- S is the set of all feasible solutions,
- C is a set of constraints that each solution must satisfy, and
- $f : S \to \mathbb{R}$ is an objective function that assigns a cost or score to each solution.

The goal is to find $s^* \in S$ such that $f(s^*)$ is optimized (maximized or minimized) while satisfying all constraints in C.

In the context of quantum computing, combinatorial problems are particularly relevant because many of them can be encoded into Hamiltonians [50] and solved approximately using variational quantum algorithms (VQAs) or quantum annealing.

In this thesis, we will use MaxCut as a representative combinatorial problem because it is widely studied, easy to understand, and fits within the QAOA application range.

MaxCut Problem

The **MaxCut problem** is a classical combinatorial optimization problem defined on an undirected graph $G = (V, E)$, where V is the set of vertices and E the set of edges. The objective is to partition the

vertices into two disjoint sets S and $\bar{S}$ such that the number of edges between the sets is maximized. Equivalently, one seeks a cut that maximizes the sum of weights of edges crossing the partition in a weighted graph. MaxCut is known to be **NP-hard**, making it a challenging problem for classical algorithms on large graphs

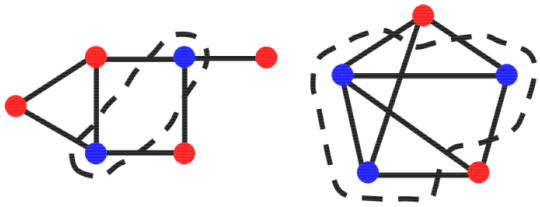


Figure 2.8: Schema of the MaxCut Problem objective. Image adapted from [51]

Mathematically, the MaxCut problem can be formulated as follows. For each vertex $i \in V$, assign a binary variable $x_i \in \{-1, +1\}$ indicating the partition of vertex i. The goal is then to maximize the cost function:

$$C_{\text{MaxCut}} = \sum_{(i,j)\in E} \frac{1 - x_i x_j}{2}.$$

Here, each edge contributes 1 to the sum if its endpoints lie in different sets and 0 if they are in the same set.

Equivalently, we can also minimize the following cost function:

$$C_{\text{MaxCut}} = \sum_{(i,j)\in E} \frac{x_i x_j - 1}{2},$$

where each edge contributes 1 to the sum if its endpoints lie in the same set and 0 if they are in different sets.

Despite its computational difficulty, MaxCut is widely used as a benchmark in both classical and quantum optimization studies due to its simplicity and clear objective function. In the context of this thesis, MaxCut serves as a representative combinatorial problem to evaluate the performance of our emulation methods. Its formulation allows straightforward mapping to QAOA, where the problem Hamiltonian encodes the cut objective.

2.3. Entanglement

Entanglement is a fundamental quantum property describing correlations between parts of a quantum system, where the state of one subsystem cannot be fully described without reference to the others. We define entangled states as those that cannot be expressed as a simple product of individual subsystem states, i.e., $\psi_1 \otimes \psi_2 \otimes \cdots \otimes \psi_n$. This means that, in general, it is not possible to assign a single state vector to any one of the n subsystems independently. Entanglement formally captures this phenomenon, which, in contrast to classical superposition, enables the construction of exponentially large superpositions while requiring only a linear amount of physical resources [52], giving us a potential computational advantage.

Consider the following state:

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|00\rangle + |01\rangle\right)$$

which can be written as a product state of two single-qubit states.

$$|\psi\rangle = |0\rangle \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Thus, ψ is not entangled. Now consider the Bell state:

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

This state cannot be written as a tensor product of two single-qubit states. Thus, we say that Φ^+ is entangled.

Entanglement can be created by quantum circuits, usually by controlled gates. In Figure 2.9 we can see how to create the Bell state from before.

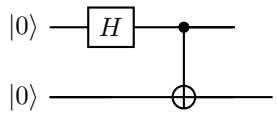


Figure 2.9: Quantum circuit creating the Bell state $\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

From:

$$H \otimes I \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix} \qquad \text{CNOT} \;=\; \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

we can verify that

$$|00\rangle \equiv \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad\Longrightarrow\quad (H\otimes I)\,|00\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\big(\,|00\rangle + |10\rangle\,\big). \tag{2.17}$$

$$\left|\Phi^+\right\rangle = \text{CNOT}\,(H\otimes I)\,|00\rangle = \text{CNOT}\left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\1\\0\end{bmatrix}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big). \tag{2.18}$$

Moreover, a multipartite quantum state is a state that involves more than two subsystems. Such states can exhibit complex entanglement patterns that are shared among several parties, going beyond pairwise correlations.

A typical example of a multipartite entangled state is the *Greenberger–Horne–Zeilinger (GHZ) state*. For three qubits, it is defined as

$$|\text{GHZ}\rangle \equiv \frac{1}{\sqrt{2}}\left(|000\rangle + |111\rangle\right),$$

where all three qubits are maximally entangled.

We have verified that quantum circuits are capable of generating entanglement. Entanglement is a fundamental resource in quantum information science: it enables nonclassical correlations that underpin quantum teleportation and superdense coding [53]. However, quantifying entanglement remains a nontrivial problem. Unlike classical correlations, entanglement cannot be fully described by a single observable or a probabilistic model. Because of this challenge, many studies have focused on developing reliable ways to detect and quantify entanglement through experimentally accessible quantities, leading to a wide range of entanglement measures.

Before delving into the topic of entanglement measures, it is useful to first clarify some concepts that will arise later.

- **Density matrix:** A quantum state can be represented by a density matrix ρ. For a composite system with subsystems A and B, the joint state is described by ρ_{AB}. Representing a state as a density matrix allows the description of both pure and mixed states, enables computation of subsystem properties via partial traces, and provides a unified framework for calculating expectation values of observables.
- **Partial Trace:** The partial trace is an operation used to extract the state of a subsystem from a larger quantum system. If a system AB is described by a density matrix ρ_{AB}, the partial trace over subsystem B is defined as:

$$\rho_A = \mathrm{Tr}_B(\rho_{AB}) = \sum_i \langle b_i|\rho_{AB}|b_i\rangle. \tag{2.19}$$

  where $|b_i\rangle$ are the elements of an orthonormal basis of subsystem B.
- **Reduced density matrix:** To study entanglement quantitatively, we often need to focus on a part of a larger quantum system. Given a composite system composed of multiple subsystems, we can describe any individual subsystem by constructing its reduced density matrix.

  The density matrix of a pure state $|\Psi\rangle_{AB}$ can be calculated by $\rho_{AB} = |\Psi\rangle_{AB}\langle\Psi|_{AB}$. Then, the state of subsystem A alone is obtained by performing a partial trace over subsystem B (see Eq 2.19). The reduced density matrix ρ_A contains all the information accessible by measurements on subsystem A.
- **Schmidt decomposition**: Any *bipartite pure state* $|\psi\rangle_{AB}$ in a Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$ can be written in the *Schmidt decomposition* form:

$$|\psi\rangle_{AB} = \sum_{i=1}^{r} \lambda_i \; |u_i\rangle_A \otimes |v_i\rangle_B \,, \tag{2.20}$$

  where:
  - $\{|u_i\rangle_A\}$ and $\{|v_i\rangle_B\}$ are orthonormal sets in $\mathcal{H}_A$ and $\mathcal{H}_B$, respectively,
  - $\lambda_i \geq 0$ are called the Schmidt coefficients satisfying $\sum_i \lambda_i^2 = 1$,
  - r is the *Schmidt rank*, i.e., the number of nonzero coefficients.

  If the bipartite state $|\psi\rangle_{AB}$ lives in $\mathcal{H}_A \otimes \mathcal{H}_B$, with

$$\dim(\mathcal{H}_A) = d_A \quad \text{and} \quad \dim(\mathcal{H}_B) = d_B,$$

  then the number of nonzero Schmidt coefficients satisfies

$$r \leq d = \min(d_A, d_B). \tag{2.21}$$

  From the Schmidt decomposition we extract some valuable information:
  - If the Schmidt rank $r = 1$, the state is separable.
  - If $r > 1$, the state is entangled.

  We say that a bipartite pure state is **maximally entangled** when all its Schmidt coefficients are equal:

$$\lambda_i = \frac{1}{\sqrt{d}} \quad \text{for all } i = 1, \ldots, d.$$

  and, in consequence, the reduced density matrix of subsystem A (or B) is

$$\rho_A = \mathrm{Tr}_B\big(\,|\psi\rangle\,\langle\psi|\,\big) = \frac{1}{d} I_d,$$

  where I_d is the $d \times d$ identity matrix.

- **Von Neumann Entropy:** The von Neumann entropy is a measure of the uncertainty encoded in the probability distribution of a quantum state. Following from the classical Shannon entropy [54]:

$$S = -\sum_i p_i \log p_i, \tag{2.22}$$

where p_i is the probability of observing the i-th outcome of a random variable. For a classical probability distribution $\{p_i\}$, the entropy is zero when there is no uncertainty, i.e., $p_1 = 1$ and all other $p_i = 0$, giving $S = 0$. Conversely, the uncertainty is maximized when all N possibilities are equally likely, $p_i = 1/N$ for each i, yielding $S = \log N$.

Similarly, for a quantum state described by a density matrix ρ, the von Neumann entropy is defined as

$$S(\rho) = -\mathrm{Tr}(\rho \log \rho). \tag{2.23}$$

Some key properties are:

  - $S(\rho) \geq 0$, and $S(\rho) = 0$ if and only if ρ is a pure state.
  - The entropy quantifies the mixedness of a subsystem: the more uniformly distributed the eigenvalues of ρ, the higher the uncertainty and the stronger the entanglement with other subsystems.

Having established the fundamental concepts of subsystems, reduced density matrices, and the Schmidt decomposition, we are now ready to discuss entanglement measures.

2.3.1. Entanglement measures

Conditions on Entanglement Measures

There exist many different entanglement measures, but all of them should satisfy some mathematical properties that capture the essential features of entanglement. A bipartite entanglement measure $E(\rho)$ is a mapping from density matrices to positive real numbers:

$$E(\rho) : \rho \mapsto \mathbb{R}^+.$$

For an entanglement measure, we mention the conditions from [55] where we identify the density matrix of a quantum state as ρ:

1. **Zero for separable states:**

$$E(\rho) = 0 \quad \text{if and only if } \rho \text{ is separable.} \tag{2.24}$$

2. **Invariance under local unitary transformations:** Entanglement should not change under local operations on the subsystems. For $U = U_A \otimes U_B$,

$$E(\rho) = E\big(U_A \otimes U_B \, \rho \, U_A^\dagger \otimes U_B^\dagger\big), \tag{2.25}$$

where the subscript indicate that the matrix is only acting on subsystem A or B.

3. **Monotonicity under LOCC:** Entanglement cannot increase under any local operations and classical communication (LOCC) Λ [14]:

$$E(\rho) \geq E(\Lambda(\rho)). \tag{2.26}$$

4. **Consistency with pure-state entanglement:** For a pure bipartite state $|\psi\rangle$, the measure reduces to the *entropy of entanglement*:

$$E(|\psi\rangle\langle\psi|) = S\big(\mathrm{Tr}_B \, |\psi\rangle\langle\psi|\,\big), \tag{2.27}$$

where $S(\rho_A)$ is the von Neumann entropy of the reduced density matrix explained below.

5. **Maximal entanglement normalization:** For the maximally entangled state $\left|\psi_d^+\right\rangle$ where d is the dimension of the system,
$$E(\left|\psi_d^+\right\rangle \left\langle\psi_d^+\right|) = \log d. \tag{2.28}$$

These conditions ensure that the entanglement measure is physically meaningful, consistent with the notion of entanglement as a resource in the quantum information field, and reduces to the standard von Neumann entropy for pure states, which will be defined below.

Entanglement measures for bipartite states

- **Von Neumann Entropy (pure states):** For a bipartite pure state $|\psi\rangle_{AB}$, the von Neumann entropy of the reduced density matrix can be seen as an entanglement metric:
$$E(|\psi\rangle\langle\psi|) = S(\rho_A) = -\mathrm{Tr}(\rho_A \log \rho_A) \quad \text{where} \quad \rho_A = \mathrm{Tr}_B(|\psi\rangle\langle\psi|). \tag{2.29}$$
- **Entanglement of Formation (EoF):** Measures the minimum average entanglement of pure states required to create a mixed state ρ using LOCC:
$$E_f(\rho) = \inf_{\{p_i, |\psi_i\rangle\}} \sum_i p_i \, S(\mathrm{Tr}_B \, |\psi_i\rangle\langle\psi_i|), \tag{2.30}$$
where the infimum is taken over all ensembles [1] $\{p_i, |\psi_i\rangle\}$ of pure states. Thus, ρ is defined as $\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i|$ and p_i is the probability of the pure state $|\psi_i\rangle$. $S(\cdot)$ is the von Neumann entropy of the reduced subsystem.
- **Entanglement Cost (E_C):** The asymptotic minimal number of maximally entangled pairs (also known as ebits) needed to create ρ via LOCC:
$$E_C(\rho) = \lim_{n\to\infty} \frac{1}{n} E_f(\rho^{\otimes n}). \tag{2.31}$$
where n represents the number of copies of state ρ needed.
- **Distillable Entanglement (E_D):** The maximal number of ebits that can be extracted from many copies of ρ using LOCC:
$$E_D(\rho) = \sup\left\{ r : \rho^{\otimes n} \xrightarrow{\text{LOCC}} (\left|\Phi^+\right\rangle\left\langle\Phi^+\right|)^{\otimes rn} \right\}. \tag{2.32}$$
where n is the number of copies considered, $|\Phi^+\rangle$ is a maximally entangled Bell pair, and r is the rate of distillation, i.e., the number of Bell pairs obtained per copy of ρ in the asymptotic limit.
- **Relative Entropy of Entanglement (E_R):** Measures the distance of ρ from the set of separable states $\mathcal{S}$:
$$E_R(\rho) = \inf_{\sigma\in\mathcal{S}} \mathrm{Tr}(\rho\log\rho - \rho\log\sigma). \tag{2.33}$$
where $\mathcal{S}$ is the set of separable states.
- **Logarithmic Negativity (E_N):** Quantifies entanglement via the partial transpose:
$$E_N(\rho) = \log \|\rho^{T_B}\|_1, \tag{2.34}$$
where ρ^{T_B} is the partial transpose with respect to subsystem B, and $\|\cdot\|_1$ is the trace norm.

Since the goal of this thesis is not to delve into the full details of entanglement, although it is an interesting subject, we refer the reader to [55]. In this work, our main focus will be on the von Neumann entropy as a measure of entanglement for bipartite pure states. The reason is that von Neumann entropy is exact, widely recognized, and easy to compute, making it the most practical choice. The other measures are either computationally harder or designed for mixed/asymptotic scenarios, which are beyond the scope of your work.

[1] An **ensemble** is a collection of quantum states $\{|\psi_i\rangle\}$ each occurring with a probability p_i, representing a mixed state $\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i|$.

Entanglement measures for multipartite states

For systems with more than two subsystems, entanglement can be more complex, but several measures try to generalize bipartite measures:

- **Global Entanglement / Meyer-Wallach Measure:** For an n-qubit pure state $|\psi\rangle$, the Meyer-Wallach measure Q is defined as

$$Q(|\psi\rangle) = 2\left(1 - \frac{1}{n}\sum_{k=1}^{n}\mathrm{Tr}(\rho_k^2)\right), \tag{2.35}$$

  where ρ_k is the reduced density matrix of the k-th qubit. Q ranges from 0 (fully separable) to 1 (maximally entangled).

- **Geometric Measure of Entanglement:** For a pure state $|\psi\rangle$, the geometric measure is

$$E_G(|\psi\rangle) = 1 - \max_{|\phi\rangle\in\mathcal{S}} |\langle\phi|\psi\rangle|^2, \tag{2.36}$$

  where the maximum is over $\mathcal{S}$, the set of fully separable states $|\phi\rangle$. It quantifies the distance from the closest separable state.

- **Multipartite Negativity:** Given a partition $A|B$ of the system, the negativity generalizes as

$$\mathcal{N}_{A|B}(\rho) = \frac{\|\rho^{T_A}\|_1 - 1}{2}, \tag{2.37}$$

  where ρ^{T_A} is the partial transpose with respect to subsystem A. Multipartite negativity can be defined by averaging over different bipartitions.

These measures are generally more difficult to compute than bipartite measures, and their interpretation can depend on the chosen partitioning of the system.

2.4. Quantum Emulators

Quantum emulators are classical computational tools designed to reproduce the behavior of computers executing quantum circuits. Early emulators primarily relied on the statevector approach, which explicitly represents the full wavefunction of an n-qubit system, requiring 2^n complex amplitudes [14]. While enabling exact results, this method is limited by exponential memory and computational requirements, making large-scale emulations infeasible on conventional computers. To overcome these limitations, several emulator paradigms have been developed, some of which are presented in the following sections and summarize in Table 2.1.

Despite limitations, quantum emulators play a crucial role in the development and testing of quantum algorithms, allowing researchers to validate circuit designs, benchmark algorithm performance, and explore error mitigation strategies before executing them on real quantum hardware. This is particularly important in the current NISQ (Noisy Intermediate-Scale Quantum) era, where quantum processors have a limited number of qubits and are prone to noise and decoherence.

2.4.1. Statevector-Based Emulators

Statevector emulators represent quantum states as vectors of complex amplitudes, evolving deterministically under unitary transformations corresponding to quantum gates [56, 57, 58]. This approach provides exact results, making it the standard reference for algorithm development and testing. Practically, gates are applied via matrix-vector multiplications, ensuring that the state remains normalized. However, memory requirements scale exponentially with the number of qubits; for instance, emulating 44 qubits in double precision requires roughly $2^{44} \approx 1.76 \times 10^{13}$ amplitudes, corresponding to about 256 TB of memory [59]. Despite these limitations, statevector emulators remain widely used for small- to medium-scale emulations due to their accuracy and simplicity.

2.4.2. Density Matrix-Based Emulators

Density matrix emulators extend statevectors to represent both pure and mixed states, allowing the inclusion of noise and decoherence. Memory requirements scale as 2^{2n} for n qubits, which limits even more the size of circuits that can be emulated. Sparsity techniques can reduce memory use, but systems with more than ~ 50 qubits remain difficult to emulate classically [56].

2.4.3. Tensor Network Emulators

Tensor network emulators map quantum circuits onto networks of tensors representing quantum states and gates [19, 60]. TN methods such as MPS, Projected Entangled Pair States (PEPS), and Tree Tensor Networks (TTNs) exploit structured patterns of entanglement to represent states efficiently. While they trade accuracy for scalability, they allow the emulation of larger systems than statevector methods. TN emulators are especially useful for condensed matter problems, specific instances of variational algorithms, and certain quantum chemistry applications [61, 12].

2.4.4. Clifford/Stabilizer Emulators

Clifford or stabilizer emulators efficiently emulate circuits composed solely of Clifford gates (Hadamard, Phase, CNOT) by tracking stabilizer states [62, 63]. They drastically reduce computational and memory requirements compared to full statevector emulations. Although limited to a subset of quantum circuits, they are invaluable for testing quantum error-correction schemes, benchmarking hardware, and studying circuits with highly structured stabilizer states.

2.4.5. Decision Diagram-Based Emulators

Decision diagram emulators compress quantum states and operations using graphical structures that exploit redundancy in amplitude patterns. Algorithms such as Quantum Multiple-valued Decision Diagrams (QMDD) and Local Invertible Map Decision Diagrams (LIMDD) can efficiently emulate circuits with repeated or partially entangled structures [64, 65]. These methods allow handling larger systems than statevectors or density matrices when circuits have regular or sparse entanglement.

2.4.6. Quantum Annealing Emulators

Simulated Quantum Annealers (SQA) approximate quantum annealing processes on classical hardware using Monte Carlo techniques like Path Integral Monte Carlo (PIMC). They capture key quantum effects such as tunneling and adiabatic evolution, making them suitable for benchmarking quantum annealers and exploring combinatorial optimization problems [66, 67]. While highly scalable in terms of qubits, SQA provides approximate results that depend sensitively on emulation parameters.

Table 2.1: Summary of Quantum Emulator Paradigms

Emulator	**Features**	**Limitations**
Statevector	Exact, deterministic	Exponential memory/time
Density Matrix	Mixed states (noisy)	Scales as 2^{2n}
Tensor Networks	Efficient for structured entanglement	Approximate; limited for deep circuits
Clifford/Stabilizer	Fast for Clifford circuits	Cannot emulate non-Clifford gates
Decision Diagrams	Compresses redundant patterns	Depends on circuit structure
Quantum Annealing	Approximates QA	Approximate; parameter-dependent

2.5. Noisy Emulations

Emulating quantum circuits under realistic noise needs to be done to understand the behavior of near-term quantum devices. Statevector emulators, while exact for ideal circuits, fail to capture decoherence, gate errors, and other imperfections present in physical hardware. To bridge this gap, noise needs to be incorporated into emulated quantum circuits, allowing benchmarking against real quantum backends [29].

To do so, noise channels mimicking the physical error processes of a target device have to be introduced into the quantum circuit. Common noise models include depolarizing noise, amplitude damping (energy relaxation), phase damping (decoherence), and measurement or initialization errors (SPAM)

[29]. These error models are illustrated in Figure 2.10.

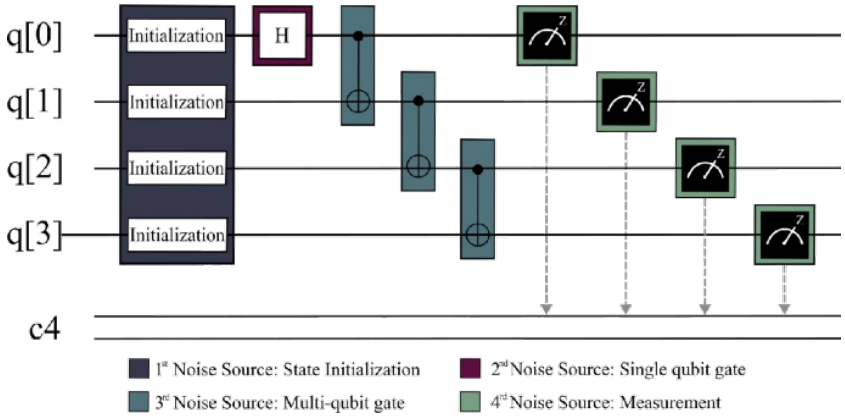


Figure 2.10: Different types of noise models applied to a quantum circuit. Boxes represents SPAM and gate noise. Figure adapted from [29].

For example, the **depolarizing channel** provides a simple model of qubit decoherence. It is defined as

$$\mathcal{E}_{\text{dep}}(\rho) = (1-p)\,\rho + \frac{p}{3}\left(X\rho X + Y\rho Y + Z\rho Z\right),$$

where X, Y, Z are the Pauli matrices and p is the depolarizing probability. Physically, the qubit remains unchanged with probability $1-p$, while with probability p an error occurs. The error may be a bit-flip, a phase-flip, or both, with each occurring with equal probability $p/3$ [68]. This model captures stochastic errors in quantum hardware and allows emulators to assess the robustness of quantum algorithms under realistic noise. For multi-qubit systems, this channel can be applied independently to each qubit or collectively to emulate global decoherence effects.

These noise channels are parameterized using calibration data obtained from the actual hardware, such as gate fidelities and qubit coherence times, to ensure realistic emulation.

The quality of noisy emulations is typically quantified using fidelity metrics, which compare the output states of the noisy emulator to either ideal statevectors [69].

During the development of the thesis we will implement some models of noise present in [29] so we can observe how the emulator behaves when running variational quantum circuits and how it scales in the presence of noise.

2.6. Related work

Several studies in the literature have focused on benchmarking and evaluating quantum circuit emulators. Some works provide comparisons between emulators of the same type but developed by different vendors such as [70] with MPS, while others benchmark quantum devices and emulators under noisy conditions or assess their individual performance [71, 29]. These efforts have contributed valuable insights into the strengths and limitations of specific emulation strategies.

However, to the best of our knowledge, there are no systematic studies directly comparing the performance of statevector-based and MPS–based emulators across different instances of quantum circuits. This constitutes the primary focus of our work.

In addition, our study extends beyond previous benchmarks by analyzing the performance of QAOA and RQAOA on various graph structures (not limited to complete graphs [27]) using statevector emulation. We further investigate the relationship between circuit entanglement, MPS bond dimension, and emulation fidelity [72] (to be described 3), providing a more comprehensive picture of the trade-offs between different emulator strategies.

3

Development Environment

3.1. Qaptiva Appliance

The *Qaptiva Appliance* is a dedicated on-premises computing system that provides a full-stack programming and emulation environment for quantum algorithm development, optimisation, and testing, as well as possible execution on quantum hardware. It offers a unified platform in which quantum applications can be programmed, compiled, optimised, emulated, and deployed on QPUs. Supporting multiple quantum computing paradigms—including gate-based, annealing, and analog approaches—the appliance is designed to be hardware-agnostic, allowing programs to target different backend technologies with minimal modification.

In practice, the Qaptiva Appliance enables users to:

- design, test, and analyse quantum algorithms within a controlled computational environment;
- study algorithmic behaviour through classical emulation, including assessing performance under different emulator paradigms;
- perform circuit optimisation and compilation tailored to specific hardware constraints, such as connectivity and noise characteristics.

3.1.1. Functional Categories

The Qaptiva framework allows some functional features which are relevant to this thesis and that can be summarised as follows.

Programming and Optimisation

The environment provides a programming interface supporting standard quantum assembly languages such as QASM, as well as high-level libraries like PyQASM, Python integration, and Jupyter notebook support. The toolchain enables circuits to be transformed and compiled using pattern-based optimisers, topology mapping, and noise-aware compilation strategies.

Emulation

There are several emulation backends available in the appliance, covering all three major quantum-computing paradigms: digital, annealing, and analog. However, in this work we focus on three primary emulation modes: the statevector emulator, which adopts full wavefunction simulation via exact linear-algebraic methods; the MPS emulator, which relies on tensor-network techniques based on matrix product state representations; and the noisy emulator, which incorporates stochastic and deterministic noise models using density-matrix simulation techniques. Other emulation frameworks, such as SQA and Analog emulation, are also available but they will not be considered here.

Execution on QPU and Hybrid Workflows

Once a circuit has been developed and optimised, Qaptiva supports hybrid workflows that combine classical and quantum resources. The same code can be executed either through emulation or on actual

QPU hardware from other vendors, enabling performance comparisons and validation of algorithmic results.

3.1.2. Hardware Specifications (Qaptiva 802)

For this study, the experiments were conducted specifically on the Qaptiva 802 model. its hardware specifications are:

- CPU (without GPU): 2 sockets, each with 28 cores at 2.0 GHz, 250 W TDP per socket.
- CPU (with GPU): 2 sockets, each with 20 cores at 1.9 GHz, 205 W TDP per socket.
- Memory: 2 TB RAM.
- Disk: 1.92 TB.

The appliance is engineered to provide a scalable classical backend for quantum emulation workloads consistent with software environment for algorithm development.

3.1.3. Software Specifications (Qaptiva 802)

All emulations reported in this work were performed using the Qaptiva 802 system with software version **1.10.2**. The complete information about the appliance characteristics and features can be found [17]. The Qaptiva environment provides both a terminal for running Python scripts and an interactive interface via Jupyter notebooks. Within this environment, we made used of several built-in libraries included with the Qaptiva software, as well as commonly used Python libraries for graph manipulation, numerical computation, data handling and optimization, including `networkx` (v3.3), `numpy` (v1.26.4), `matplotlib` (v3.9.0), `pandas` (v2.2.2) and `scipy` (v1.13.1). Standard Python libraries such as `os`, `time`, `json`, and `csv` were also used (Python 3.12.3).

In addition, the software supports *plugins*, modular components that can be inserted into the quantum execution workflow to transform, optimize, or analyze circuits before and after their execution.

Plugins

Each plugin acts as a processing layer that either pre-processes the quantum job (e.g., circuit rewriting, gate fusion, or mapping) or post-processes the results (e.g., observable extraction, optimization feedback). These plugins are connected in a stacked architecture, meaning that a circuit submitted to a QPU or emulator passes sequentially through the configured plugins of the stack.

It can be represented as follows.

$$\text{plugin}_1 \,|\, \text{plugin}_2 \,|\, \ldots \,|\, \text{QPU}$$

where each plugin modifies the input or output data stream according to its specific role.

Beyond the built-in plugins, additional plugins can be explicitly inserted to optimize performance in specific contexts. In this work, we only used some of the available plugins, selected for their direct relevance to the implemented workflows.

FusionPlugin The `FusionPlugin` is a circuit-rewriting plugin designed to group adjacent gates into larger composite gates, forming an equivalent circuit that contains fewer but more complex operations. By reducing the total number of gate applications, this plugin minimizes the number of matrix multiplications required during emulations, improving computational efficiency.

The behavior of the `FusionPlugin` can be tuned through a parameter specifying the fusion strategy to be employed, which includes three distinct strategies:

- `naive strategy`: Gates are accumulated sequentially in the order they appear in the circuit.
- `adjacent strategy`: A reference gate of a given size is selected, and all gates located to its left and right are merged into a single operation.
- `eager strategy`: It maintains a collection of gate groups and attempts to merge each incoming gate into one of these existing groups. It is the default option, which we have kept.

CircuitInliner The `CircuitInliner` plugin operates by expanding composite or parameterized subcircuits into a single flat circuit. This transformation simplifies circuit hierarchies, ensuring compatibility with emulators such as the `MPS` emulator.

Nnizer The `Nnizer` plugin addresses the *qubit connectivity* constraints that arise when mapping logical circuits onto physical architectures. This is relevant in our case when using the `MPS` emulator, which requires qubits to be arranged in a linear topology, i.e. gates can only be applied to consecutive qubits.

The `Nnizer` plugin performs the mapping by inserting the minimum number of `SWAP` operations required to preserve logical equivalence.

The plugin provides several algorithms to perform this nnization process:

- `atos`: Atos algorithm (a generalization of the method introduced by Hirata *et al.* [73]). This is the default algorithm and is optimized for performance and integration with the Atos emulation backends.
- `sabre`: Implementation of the SABRE (Swap-Based Bidirectional Heuristic Search) algorithm proposed by Li, Ding, and Xue [74].
- `bka`: Best Known Algorithm (BKA), based on the approach of Zulehner, Paler, and Wille [75].
- `pbn`: Pattern-Based Optimizer, a generalization of the technique described by Shafaei, Wille, and Drechsler [76].

ScipyMinimizePlugin The `ScipyMinimizePlugin` provides a bridge between the Qaptiva environment and the optimization routines available in the *SciPy* library. It is designed to support hybrid quantum-classical workflows such as the VQE or QAOA. Although such a plugin would be highly convenient in the context of this research, it was not compatible with the `MPS` paradigm employed in the current work, as the required cost-function feedback loop is not supported by this emulation backend. So the optimization loop has been manually implemented with the `scipy.optimize.minimize` function.

Some plugins are automatically integrated in the emulation backend to enhance its internal performance. For example, the `FusionPlugin` is enabled by default in the `LinAlg` emulator. Similarly, the `CircuitInliner` is used by the `MPS` emulator when initialized.

3.1.4. Accessing the Qaptiva Appliance

All interactions with the Qaptiva 802 appliance are performed remotely from a client machine. Access is established via a Virtual Private Network (VPN) using Cisco software [77], which provides a secure connection to the appliance network. After the VPN connection is active, one can connect to the appliance through a secure shell (SSH) session. In this work, SSH connections are managed using *MobaXterm* [78], which provides an integrated terminal and session management for remote command execution (Figure 3.1). Then the command `./launch_qlm_notebooks` is executed, which launches the working environment in a web browser, as illustrated in Figure 3.2.

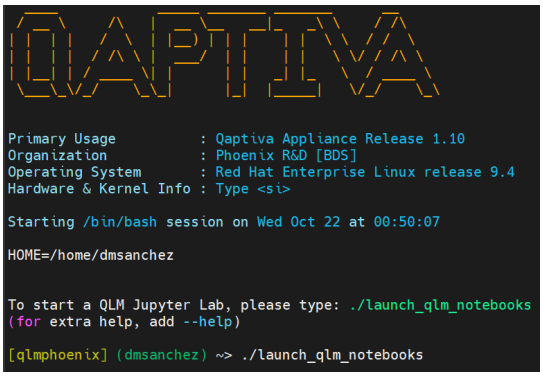


Figure 3.1: SSH session to the Qaptiva Appliance using MobaXterm.

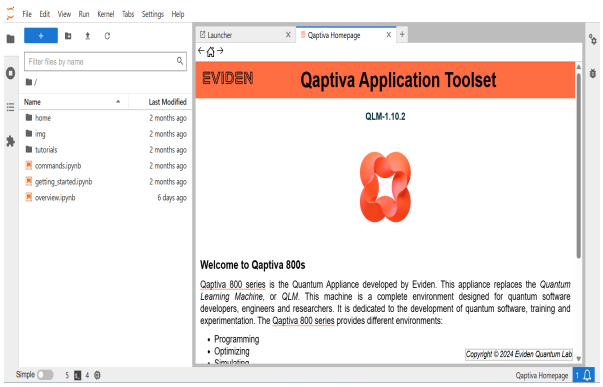


Figure 3.2: Jupyter notebook environment on the Qaptiva Appliance.

3.2. Selected Emulators

Among the various emulators available in the environment, this work focuses on LinAlg, NoisyQProc, and MPS. These three emulators have been selected due to their usefulness and complementary characteristics, and they will be described in more detail below.

3.2.1. LinAlg Emulator

The *LinAlg* Emulator is the statevector-based emulator provided by the Qaptiva Appliance. It represents the quantum state of the system as a complex vector and emulates the evolution of this state through **deterministic** linear algebra operations. In this emulator, each quantum gate is expressed as a unitary matrix, and its action on the statevector is realized through matrix–vector multiplication. To understand in detail how the emulator performs these operations, it is necessary to introduce two mathematical tools: the *Kronecker product* and the *qubit-wise multiplication*.

Kronecker product.

The Kronecker product (or tensor product) is a fundamental operation in linear algebra that combines two matrices into a larger block-structured matrix. Given a matrix $A \in \mathbb{C}^{m\times n}$ and a matrix $B \in \mathbb{C}^{p\times q}$, their Kronecker product is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix},$$

which results in a matrix of size $(mp) \times (nq)$.

As an example, we can take matrices $A, B \in \mathbb{C}^{2\times 2}$

$$A \otimes B = \begin{bmatrix} a_{00}B & a_{01}B \\ a_{10}B & a_{11}B \end{bmatrix} = \begin{bmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{01}b_{00} & a_{01}b_{01} \\ a_{00}b_{10} & a_{00}b_{11} & a_{01}b_{10} & a_{01}b_{11} \\ a_{10}b_{00} & a_{10}b_{01} & a_{11}b_{00} & a_{11}b_{01} \\ a_{10}b_{10} & a_{10}b_{11} & a_{11}b_{10} & a_{11}b_{11} \end{bmatrix}.$$

which results in a 4×4 matrix.

In the context of quantum computing, the Kronecker product provides the mathematical foundation for constructing multi-qubit states and operators from single-qubit components [14].

We can also use the Kronecker product to express the statevector of a multi-qubit system. Each qubit is represented in the computational basis by a two-dimensional complex vector:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

For instance, the initial state $|00\rangle$ of a two-qubit system can be expressed as the tensor product of two single-qubit states:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Likewise, the state $|10\rangle$ corresponds to

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Thus, we can represent the following state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

in vector form as

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

So the Kronecker product provides a way to construct the full statevector of a multi-qubit system from the tensor products of individual qubit states.

Qubit-wise multiplication.

Quantum circuits are evaluated through a sequence of matrix–vector multiplications, thus, it is logical to represent each layer of gates as a single large matrix acting on the global statevector. However, as the number of qubits increases, the size of the matrices generated via the Kronecker product grows exponentially, with dimension $2^n \times 2^n$ for an n-qubit system. In practice, these matrices are often mostly sparse, particularly in layers where only a few gates are applied.

This sparsity indicates that we can achieve significantly better time and memory performance by avoiding the explicit storage of these large matrices, as explained in [79] and shown in Figure 3.3.

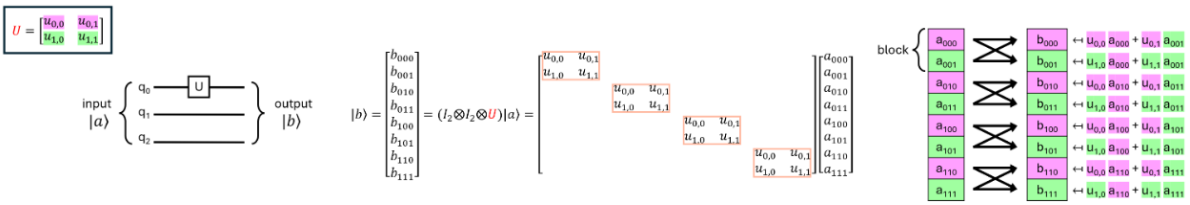


Figure 3.3: Efficient Qubit-Wise Multiplication. Image taken from [79].

Example

To exemplify who the LinAlg emulator works, we will make use of the circuit in 2.4 but we will get rid off the measurement step in order to show the final statevector output:

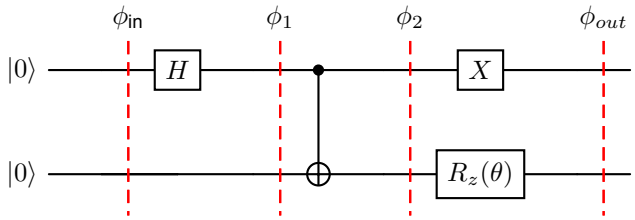


Figure 3.4: Two-qubit quantum circuit example illustrating the state evolution in the LinAlg emulator. The initial state is ϕ_{in}, and intermediate states ϕ_1, ϕ_2, ϕ_{out} correspond to the statevector after each operation.

Let the initial state be

$$\phi_{\text{in}} = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Hadamard gate on qubit 0: After applying the Hadamard gate H to the first qubit, expanded to the full two-qubit space as $H \otimes I$, the statevector becomes

$$\phi_1 = (H \otimes I)\, \phi_{\text{in}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

CNOT gate: The state after the CNOT gate is

$$\phi_2 = \mathsf{CNOT}\,\phi_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}.$$

Pauli-X gate on qubit 0 and $R_z(\theta)$ gate on qubit 1: Applying $X \otimes I$ to the first qubit followed by $I \otimes R_z(\theta)$ on the second qubit transforms the state ϕ_2 into

$$\phi_{\mathsf{out}} = (I \otimes R_z(\theta))\,(X \otimes I)\,\phi_2 = (X \otimes R_z(\theta))\,\phi_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\e^{i\theta/2}\\e^{-i\theta/2}\\0\end{bmatrix}.$$

Thus, the final statevector output ϕ_{out}, which can be expressed in the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ as

$$\phi_{\mathsf{out}} = 0 \cdot |00\rangle + \frac{e^{i\theta/2}}{\sqrt{2}}\,|01\rangle + \frac{e^{-i\theta/2}}{\sqrt{2}}\,|10\rangle + 0 \cdot |11\rangle.$$

If we consider the measurement probabilities, we find

$$P(|01\rangle) = \left|\frac{e^{i\theta/2}}{\sqrt{2}}\right|^2 = \frac{1}{2}, \qquad P(|10\rangle) = \left|\frac{e^{-i\theta/2}}{\sqrt{2}}\right|^2 = \frac{1}{2},$$

and

$$P(|00\rangle) = P(|11\rangle) = 0.$$

The key advantage of the LinAlg emulator is that it provides the complete representation of the state, allowing access to the amplitudes of all basis states simultaneously. In contrast, a real quantum processor (QPU) only returns a single classical outcome per shot, which is sampled according to these probabilities. Thus, the emulator enables deterministic verification and detailed analysis of the quantum state that would not be directly observable on physical hardware.

3.2.2. NoisyQProc Emulator

The NoisyQProc Emulator extends statevector emulations to include the effects of quantum noise, allowing the study of errors present in quantum circuits. This emulator supports two distinct approaches for noise modeling: deterministic and stochastic modes.

Deterministic mode.

In deterministic mode, the emulator explicitly represents the system using a *density matrix*. Noisy gate operations are emulated by directly manipulating this matrix, providing an exact and reproducible description of the quantum evolution under noise. Due to the quadratic scaling of the density matrix, this approach is limited to relatively small numbers of qubits. Deterministic emulations are therefore most suitable for small circuits where precise modeling of all noise contributions is required.

Stochastic mode.

The stochastic mode represents the quantum system as a collection of *pure states* sampled according to a probability distribution derived from the density matrix:

$$\rho = \sum_j p_j |\psi_j\rangle\langle\psi_j|,$$

where ρ is the density matrix, p_j is the probability associated with the pure state $|\psi_j\rangle$. Each sampled state evolves under the same unitary transformations used in ideal emulations. The final outcome is obtained by averaging over all samples, which introduces statistical uncertainty but allows emulation of a larger number of qubits without exceeding memory limits. The trade-off is that a sufficiently large number of samples must be collected to obtain statistically meaningful results, which increases computation time [80]. Consequently, stochastic emulations are preferred for larger circuits, provided that the number of samples is chosen to balance accuracy and runtime.

Error Implementation in NoisyQProc

In the emulations, we included the Depolarizing Channel to model standard quantum noise, as previously described in chapter 2. In addition to depolarization, we incorporate time errors, which account for the execution duration of specific quantum operations. For this purpose, we assign execution times to key gates and processes, including CNOT, H, R_X, PH, state preparation, and measurements. This approach allows the emulator to capture timing-dependent effects that may influence the evolution of the quantum state, providing a more realistic emulation of noisy quantum circuits.

By combining standard depolarizing errors with these time-dependent contributions, the NoisyQProc emulator can model both intrinsic gate noise and temporal aspects of quantum operations, which is essential in modelling real world behavior of current quantum devices.

3.2.3. Matrix Product State Emulator

During this work, we have made us of the *Matrix Product State* emulator available in the Qaptiva Appliance. This emulator efficiently emulates quantum systems characterized by low entanglement, and it provides tunable parameters—as the *bond dimension*—that directly affect both accuracy and performance. To understand the influence of this parameter, we first introduce the theoretical foundations of the MPS formalism.

Foundations of MPS

The MPS representation stems from the *Density Matrix Renormalization Group* (DMRG) method [81], a variational algorithm to determine ground states of one-dimensional quantum systems. DMRG constructs an optimized wavefunction by iteratively retaining the most relevant components of the system's density matrix. The resulting wavefunction can be expressed as an MPS. Thus, the MPS representation provides a compact representation of many-body quantum states when entanglement is limited. While a general n-qubit state requires 2^n complex amplitudes, an MPS captures the same information through a sequence of local tensors connected by internal *bond indices*, whose dimension limits the amount of entanglement that can be represented.

Consider a system of n qubits with computational basis states $|i_1 i_2 \dots i_n\rangle$, where $i_k \in \{0, 1\}$. Remember that a general pure state can be written as

$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} c_{i_1 i_2 \dots i_n} |i_1 i_2 \dots i_n\rangle . \tag{3.1}$$

The MPS representation expresses the coefficients $c_{i_1 i_2 \dots i_n}$ as a product of local tensors

$$c_{i_1 i_2 \dots i_n} = A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n}, \tag{3.2}$$

where each $A^{[k]}_{i_k}$ is a matrix. For open boundary conditions (see Figure 3.5 to observe the difference with periodic boundary conditions), the state takes the form

$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n} |i_1 i_2 \dots i_n\rangle . \tag{3.3}$$

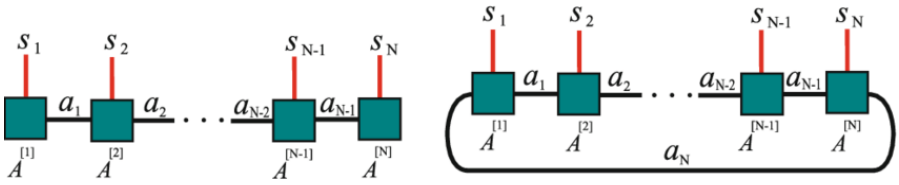


Figure 3.5: Representation for MPS with Open Boundary Conditions (left) and Periodic Boundary Conditions (right). Here s_i are the physical indexes and a_i are the virtual or bond indexes. Image taken from [82].

Here, $A^{[k]}_{i_k}$ is a matrix of dimension $D_{k-1} \times D_k$, where D_k denotes the *bond dimension* at site k. The parameters D_k quantify the maximum entanglement that can be captured between subsystems divided

at bond k. For open boundary conditions, $D_0 = D_n = 1$, so at the boundaries,$A^{[1]}$ and $A^{[n]}$ are row and column vectors, respectively.

Focusing on a specific MPS tensor $A^{[k]}_{i_k}$, we distinguish two types of indices:

- **Physical index** i_k (s_k in Figure 3.5): This corresponds to the *local Hilbert space* of the qubit at site k, i.e., the computational basis states $|0\rangle$ and $|1\rangle$. Its dimension represents the local Hilbert space dimension d (for qubits, $d = 2$).
- **Bond (or virtual) indices** α_{k-1}, α_k (a_k in Figure 3.5): These connect neighboring tensors in the chain. Their dimension D_k, called the *bond dimension*, determines the size of the matrices $A^{[k]}_{i_k}$ at that site.

Before going through an example of the MPS decomposition, we need first to introduce the last ingredient:

Singular Value Decomposition (SVD) Given an arbitrary matrix $M \in \mathbb{C}^{m\times n}$, its Singular Value Decomposition (SVD) factorizes it as

$$M = USV^\dagger, \tag{3.4}$$

where:

- $U \in \mathbb{C}^{m\times r}$ is the unitary matrix of left singular vectors,
- $V \in \mathbb{C}^{n\times r}$ is the unitary matrix of right singular vectors,
- $S \in \mathbb{R}^{r\times r}$ is diagonal with non-negative singular values $s_1 \geq s_2 \geq \cdots \geq s_r \geq 0$,
- $r = \mathrm{rank}(M)$ is the rank of M.

The SVD is important because it provides a canonical way to decompose a matrix into orthogonal components, separating its dominant directions (given by large singular values) from less significant ones.

Now we can procced to exemplify how MPS decompose and represent qubit systems:

Example on a 3-qubit system: Consider a system of three qubits. A general pure state can be written as

$$|\psi\rangle = \sum_{i_1,i_2,i_3=0}^{1} c_{i_1 i_2 i_3} \, |i_1 i_2 i_3\rangle \, , \tag{3.5}$$

where the coefficients $c_{i_1 i_2 i_3}$ form a rank-3 tensor representing the amplitudes in the computational basis.

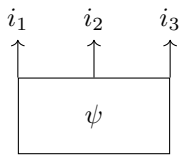


Figure 3.6: Graphical representation of a general three-qubit pure state $|\psi\rangle$.

The goal now is to represent this quantum state in smaller tensors by iteratively applying the SVD.

Step 1: First SVD Decomposition We start by separating the first qubit from the rest. Reshape the coefficient tensor $\psi_{i_1 i_2 i_3}$ into a matrix with rows labeled by i_1 and columns by $(i_2 i_3)$:

$$C_{i_1,(i_2 i_3)} = \psi_{i_1 i_2 i_3}. \tag{3.6}$$

Perform an SVD of this matrix:

$$C_{i_1,(i_2 i_3)} = \sum_{\alpha_1} U^{[1]}_{i_1,\alpha_1} S^{[1]}_{\alpha_1} (V^{[1]\dagger})_{\alpha_1,(i_2 i_3)}. \tag{3.7}$$

We can then define

$$A^{[1]}_{i_1,\alpha_1} = U^{[1]}_{i_1,\alpha_1}, \quad \tilde{\psi}_{\alpha_1 i_2 i_3} = S^{[1]}_{\alpha_1}(V^{[1]\dagger})_{\alpha_1,(i_2 i_3)}, \tag{3.8}$$

so that

$$\psi_{i_1 i_2 i_3} = \sum_{\alpha_1} A^{[1]}_{i_1,\alpha_1}\, \tilde{\psi}_{\alpha_1 i_2 i_3}. \tag{3.9}$$

This expresses the original three-qubit tensor as a contraction of two smaller tensors: $A^{[1]}$ (associated with qubit 1) and $\tilde{\psi}$ (containing qubits 2 and 3).

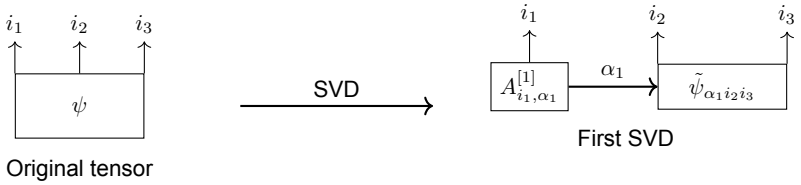


Figure 3.7: First step of the MPS decomposition: the original three-qubit tensor $\psi_{i_1 i_2 i_3}$ (left) is split into two tensors $A^{[1]}$ and $\tilde{\psi}$ (right) by performing an SVD on the first qubit. The double line represents the bond index α_1.

Step 2: Second SVD Decomposition Next, we separate qubit 2 from qubit 3 in the tensor $\tilde{\psi}_{\alpha_1 i_2 i_3}$. Reshape it as a matrix with rows labeled by (α_1, i_2) and columns by i_3:

$$C_{(\alpha_1 i_2),i_3} = \tilde{\psi}_{\alpha_1 i_2 i_3}. \tag{3.10}$$

Perform an SVD:

$$C_{(\alpha_1 i_2),i_3} = \sum_{\alpha_2} U^{[2]}_{(\alpha_1 i_2),\alpha_2} S^{[2]}_{\alpha_2} (V^{[2]\dagger})_{\alpha_2,i_3}. \tag{3.11}$$

Define the tensors:

$$A^{[2]}_{i_2,\alpha_1\alpha_2} = U^{[2]}_{(\alpha_1 i_2),\alpha_2}, \quad A^{[3]}_{i_3,\alpha_2} = S^{[2]}_{\alpha_2}(V^{[2]\dagger})_{\alpha_2,i_3}. \tag{3.12}$$

Now the original tensor can be written as a contraction of three smaller tensors:

$$\psi_{i_1 i_2 i_3} = \sum_{\alpha_1,\alpha_2} A^{[1]}_{i_1,\alpha_1} A^{[2]}_{i_2,\alpha_1\alpha_2} A^{[3]}_{i_3,\alpha_2}. \tag{3.13}$$

The resulting final MPS representation for the three-qubit state can be seen in Figure 3.8.

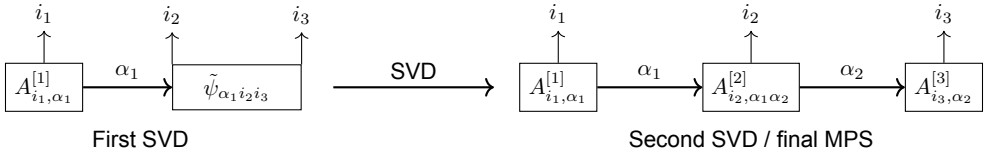


Figure 3.8: Representation of the MPS decomposition: the first SVD (left) splits the three-qubit tensor into $A^{[1]}$ and $\tilde{\psi}$, and the second SVD (right) splits $\tilde{\psi}$ into $A^{[2]}$ and $A^{[3]}$, yielding the final MPS.

Truncation and Bond Dimension in MPS

In the previous three-qubit example, we performed a full decomposition of the state using successive SVDs. In practice, however, it is often unnecessary to retain all singular values. The SVD naturally orders the singular values from largest to smallest, meaning that the largest singular values encode the most significant correlations in the system.

If we allow the bond dimension to be as large as $\chi = 2^{n/2}$ for a system of n qubits, the MPS can represent *any* quantum state exactly, recovering the full Hilbert space without truncation. By keeping only the largest χ singular values at each bond, we can obtain an *approximate* but much more compact

representation of the quantum state. This truncation reduces both memory requirements and computational cost while preserving the most relevant features of the state. In the Qaptiva Appliance, this approximation is controlled by the `bond_dimension` parameter, which sets a maximum χ that applies uniformly to all bonds in the MPS. Individual bond dimensions cannot be set independently.

Computational Complexity The advantage of MPS truncation is most evident when comparing the computational cost with that of a full state-vector representation:

- **Full representation:** Storing or manipulating a generic n-qubit state requires $O(2^n)$ memory and $O(2^n)$ computational cost for basic operations.
- **MPS with bond dimension χ:** By truncating to a maximum bond dimension χ, the storage and computation scale as

$$O(nd\chi^2) \quad \text{(memory)}, \quad and \quad O(nd^3\chi^3) \quad \text{(operations)}, \tag{3.14}$$

  where d is the local Hilbert space dimension (2 for qubits) and n is the number of sites.

This reduction in complexity is what allows MPS-based emulators to emulate some types of quantum circuits efficiently, even for relatively large numbers of qubits.

Applying Quantum Gates to MPS

One of the main advantages of representing a quantum state as an MPS is that quantum gates can be applied locally.

Single-Qubit Gates A single-qubit gate U acting on qubit k modifies only the tensor $A^{[k]}$. Mathematically, if $A^{[k]}_{i_k,\alpha_{k-1}\alpha_k}$ is the MPS tensor at site k, the updated tensor after applying U is

$$\tilde{A}^{[k]}_{i_k,\alpha_{k-1}\alpha_k} = \sum_{j_k=0}^{d-1} U_{i_k j_k} A^{[k]}_{j_k,\alpha_{k-1}\alpha_k}. \tag{3.15}$$

Only the physical index of that tensor is transformed; bond indices remain unchanged.

Two-Qubit Gates For a two-qubit gate G acting on neighboring qubits k and $k+1$, the operation involves contracting the two tensors with the gate:

$$\tilde{A}^{[k,k+1]}_{i_k i_{k+1},\alpha_{k-1}\alpha_{k+1}} = \sum_{j_k,j_{k+1}=0}^{d-1} G_{i_k i_{k+1},j_k j_{k+1}} A^{[k]}_{j_k,\alpha_{k-1}\alpha_k} A^{[k+1]}_{j_{k+1},\alpha_k\alpha_{k+1}}. \tag{3.16}$$

After this contraction, the result is reshaped as a matrix and an SVD is performed to split it back into two MPS tensors. We show an example on how gates are applying in the MPS representation in Figure 3.9. After the last step, we need to perform an SVD to recover the original MPS representation.

Efficiency of MPS in Gate Application and Observable Computation This representation exploits the locality of operations which makes the emulation efficient. When applying a local gate—on neighboring sites— only the tensors corresponding to those sites (and the connecting bond) need to be updated, while the rest of the network remains unchanged. This locality reduces the computational cost to run the circuit.

Moreover, it allows an efficient computation of norms and expectation values of observables. These quantities can be evaluated by sequentially contracting tensors from one side of the chain, rather than over the entire Hilbert space. When the MPS is brought into its left- or right-canonical form, all tensors except one (the one associated with the observable) contract to the identity. This simplifies calculations, as illustrated in Figure 3.10, where the network contraction for the expectation value $\langle\psi|\hat{O}|\psi\rangle$ reduces to a local computation involving only the tensors at the position of $\hat{O}$.

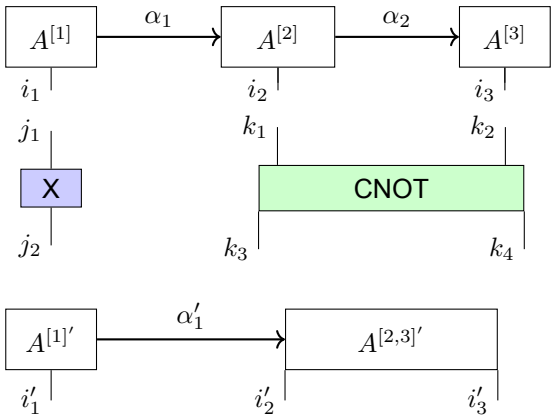


Figure 3.9: Application of an X-gate and a CNOT gate to a 3-qubit MPS. The top layer shows the original tensors, the middle layer the applied gates (X and CNOT), and the bottom layer the resulting updated tensors $A^{[1]'}$ and $A^{[2,3]'}$.

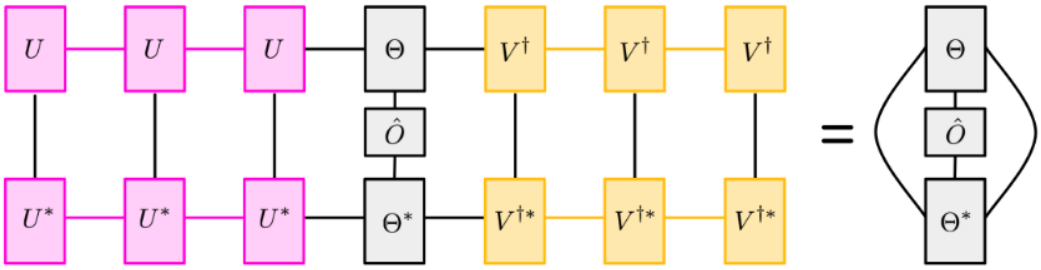


Figure 3.10: Graphical representation of the computation of an observable in an MPS. Adapted from `PennyLane` tutorial on MPS [83].

Entanglement in MPS One of the key insights underlying the efficiency of MPS is their close connection with low entanglement structures. In an MPS, the singular values associated with each bond naturally encode the bipartite entanglement between the subsystems obtained by cutting the chain at that bond. We can get the Schmidt decomposition of the full quantum state across a given bond as

$$|\psi\rangle = \sum_{\alpha=1}^{\chi} \lambda_\alpha \, |\phi_\alpha^L\rangle \otimes |\phi_\alpha^R\rangle,$$

the coefficients λ_α are the singular values obtained from the corresponding SVD in the MPS construction. The von Neumann entanglement entropy associated with this bipartition is defined as

$$S = -\sum_{\alpha=1}^{\chi} \lambda_\alpha^2 \log \lambda_\alpha^2.$$

This quantity measures the degree of entanglement between the two subsystems.

Given a fixed bond dimension χ, the maximal entanglement entropy is achieved when all singular values are equal, i.e. $\lambda_\alpha^2 = 1/\chi$. In that case,

$$S_{\max} = \log \chi.$$

Hence, the bond dimension directly limits the amount of entanglement that an MPS can represent, which is called the area law of entanglement. States obeying an *area law* for entanglement can therefore be efficiently expressed as MPS with modest χ. In one spatial dimension, the area law implies that the entanglement entropy does not scale with the total system size.

4

Proposed Solution

This chapter focuses on the description of the experimental setup and testing procedures. Detailed results, including exact plots, quantitative comparisons, and an in-depth discussion of the observed trends, are presented in the following chapter. Where appropriate, forward references are provided to guide the reader toward the corresponding results and to clarify the motivation behind specific design choices and conclusions.

4.1. Proposed Methodology

To develop and evaluate the proposed solution, a cyclic methodology was chosen. This approach suited the iterative nature of the research, allowing the work to evolve as new insights and results were discovered. It supported ongoing refinement and reassessment throughout the different stages of the project. The process was organized into five main stages: **Research**, **Implementation**, **Execution**, **Results**, and **Analysis**. Each stage informed the others, creating a continuous feedback loop that helped improve both the methodology and the experimental outcomes. A representation of this methodology is shown in Figure 4.1.

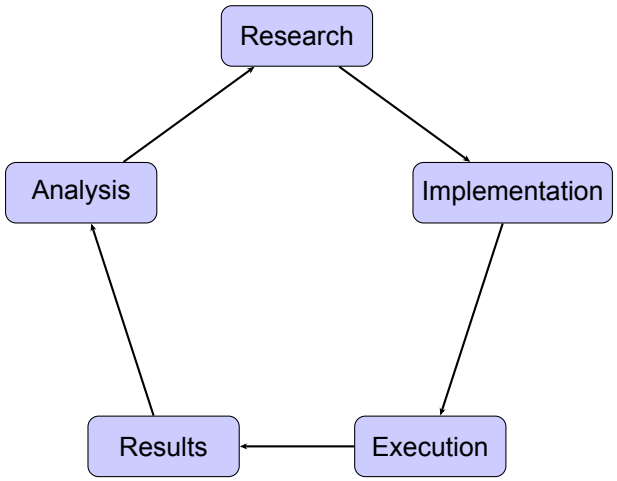


Figure 4.1: Cyclic methodology adopted in this work, showing the five stages involved.

We now explain what does each step represents:

4.1.1. Research

In the initial stage, new research questions are formulated after a new state-of-the-art review and investigated to ensure they are relevant, well-defined, and feasible within the project scope. The objective of this phase is to achieve a comprehensive understanding of the problem space and its underlying

concepts. During this step, it is also crucial to identify the tools and specific instances to be used, as well as to establish the expected outcomes and evaluation criteria. This stage provides the theoretical and conceptual foundation upon which the following stages are built.

4.1.2. Implementation

Once the research direction is established, the corresponding implementation is carried out within the Qaptiva Environment, using Python as the primary programming language. The implementation is designed to be modular and scalable using an object-oriented approach, enabling the reuse and extension of components across different experiments. This includes the automation of experiment configuration, execution, and data collection to reduce manual intervention and improve reproducibility. Furthermore, special attention is given to code organization and parameterization to facilitate systematic testing of different emulators (LinAlg, MPS and NoisyQProc).

4.1.3. Execution

This stage involves performing the actual emulations and experiments to obtain quantitative results. Depending on the circuit complexity and the emulator selected, executions can vary significantly in duration—from a few seconds for simple cases to several hours or even days for larger and emulations. During execution, various challenges may arise, such as computational errors, insufficient memory, or limited quantum processing resources. These issues are identified and resolved before proceeding, thereby ensuring the reliability of the experimental outcomes.

4.1.4. Results

Upon completion of the execution phase, the outputs are collected, processed, and stored in a structured format, typically as comma-separated value (CSV) files. This approach ensures that all generated data remain well-organized and easily accessible for further processing. The storage structure is designed to maintain traceability between experimental parameters and corresponding results, which is essential for the reproducibility and comparability of experiments. In addition, intermediate logs and performance statistics are also preserved to support detailed post-experiment analysis.

4.1.5. Analysis

The final stage focuses on interpreting the collected data and extracting the relevant performance metrics. Statistical and visual analyzes are performed to assess emulator behavior in terms of accuracy, scalability and resource consumption (mainly time). We employ plots and comparative graphs to identify trends and deviations across the different emulators. Based on these observations, conclusions are drawn regarding the effectiveness and limitations of each emulation instance. The insights obtained from this analysis often give rise to new research questions or hypotheses, thus initiating another iteration of the cyclic methodology.

4.2. First Iteration

This section details the concrete steps taken during the development and evaluation of the first approach proposed to this work, following the cyclic methodology described above.

4.2.1. Research

The first approach originated from the company's ongoing work with the Quantum Approximate Optimization Algorithm and the desire to further investigate its behavior and performance under different emulation conditions. The main research questions were focused on comparing the performance of different emulators available in the Qaptiva environment, namely LinAlg and MPS.

In this phase, we explored the theoretical background of:

- Quantum state representations on the statevector emulation formalism.
- Quantum state representations on the Matrix Product State formalism and its parameters.
- Noise modeling and the configuration of emulators to include or exclude different parameters.
- The MaxCut problem as a benchmark for QAOA performance, given its prevalence and tractability in quantum literature.

We decided to use regular graphs (specifically 2-regular cyclic graphs) as benchmark instances, since their structured nature facilitates controlled experimentation and reproducibility. Our main hypotheses were the following:

- MPS emulation will scale better for larger systems compared to the LinAlg emulator, especially in 2-regular graph instances.
- We will be able to see the first regimes where each emulator has an advantage over the other.

Additionally, we investigated the application of noise following the methodology proposed in [29]. Several types of noise were considered, including the Depolarizing Channel, Idle noise, and SPAM errors. The goal was to assess their impact on QAOA energy values and computational performance.

4.2.2. Implementation

An agnostic class structure was developed to support both MPS and LinAlg emulators within a unified framework. This class allowed switching between backends while maintaining consistent circuit generation and execution workflows.

To include noise, the implementation used Qaptiva's NoisyQProc QPU, employing an all-to-all connectivity between qubits and a deterministic emulation method based on the full density matrix. Noise parameters included gate execution times (CNOT, H, RX, PH, state preparation, and measurement), SPAM fidelities for single- and two-qubit gates and measurements, as well as T1 and T2 coherence times. The configurations for BASELINE, Target, and Desired noise settings are summarized in Table 4.1 and were taken from [84]. Here, BASELINE refers to the current noise levels observed in superconducting QPUs, Target represents the noise levels achievable through error mitigation and correction techniques, and Desired corresponds to the noise levels expected once fault-tolerant quantum computers become available.

Table 4.1: Noise parameters for the BASELINE, Target, and Desired configurations. All time values are given in nanoseconds (ns), while fidelity and error metrics are expressed as normalized values between 0 and 1.

Parameter	BASELINE	TARGET	Desired
CNOT_time	25 ns	25 ns	25 ns
H_time	25 ns	25 ns	25 ns
RX_time	25 ns	25 ns	25 ns
PH_time	25 ns	25 ns	25 ns
state_prep_time	1000 ns	1000 ns	1000 ns
meas_time	200 ns	100 ns	100 ns
t1	100000 ns	200000 ns	340000 ns
t2	100000 ns	200000 ns	340000 ns
state_prep_fid	0.98	0.99	0.99412
meas_fid	0.99	0.995	0.99706
eps_1	0.0004	0.0002	0.00012
eps_2	0.003	0.0005	0.00029

A configuration script allowed defining parameters such as optimizer, number of iterations, emulator type, fidelity threshold, and noise settings. This facilitated reproducibility and minimized manual code changes between runs.

4.2.3. Execution

The experimental phase consisted of running QAOA circuits for cyclic graphs (2-regular) with node sizes ranging from 5 to 11. For each instance, the corresponding MaxCut Hamiltonian was constructed.

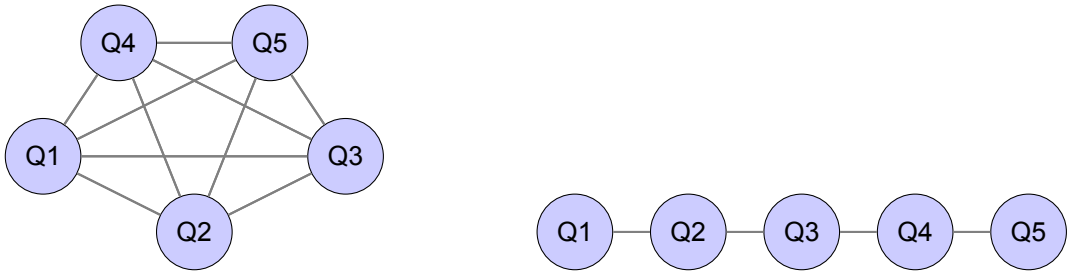
We tested two optimizers: *Nelder-Mead* and *COBYLA*, both with a limit of 200 iterations. The ansatz depth was fixed to 1 to prevent excessive computation times while still capturing basic algorithmic behavior. Every experiment was tested for the three emulators: `LinAlg`, `MPS` and `NoisyQProc`.

For the MPS emulator, instead of fixing the bond dimension, we used a dynamic fidelity-based truncation approach. A target fidelity threshold of 0.95 (as suggested in [85]) was set, allowing the bond

dimension to adapt automatically while preserving accuracy.

During execution, several issues were encountered:

- The MPS emulator was incompatible with Qaptiva's built-in `ScipyOptimizationPlugin`, as it requires non-parameterized matrices. Consequently, a custom optimization function was developed using the `minimize` method from the `scipy` library.
- MPS only supports linear topologies (neighbor-only gates), whereas Qaptiva defaults to an all-to-all (A2A) connectivity. Therefore, a custom linear topology was manually defined for MPS and replicated across all emulators for fairness. (See Fig. 4.2 for an illustration of both topologies.)



(a) All-to-All Connectivity **(b)** Linear Topology

Figure 4.2: Comparison of 5-qubit topologies. Left: All-to-All connectivity, where each qubit can interact with any other. Right: Linear topology, where qubits interact only with their nearest neighbors.

Additionally, in the first iteration we investigated the effect of noise on QAOA circuits. Multiple noise types were initially considered: Depolarizing Channel, Idle noise, and SPAM errors. Execution times with full noise models were several orders of magnitude higher than noiseless emulations, while energy values remained very similar to those obtained using the LinAlg emulator. Based on this, we simplified all subsequent experiments to include **only the Depolarizing Channel**.

The Depolarizing Channel is parameterized by:

- `eps_1`: error probability for single-qubit gates.
- `eps_2`: error probability for two-qubit gates.

Gate times for all operations (CNOT, H, RX, PH, state preparation, and measurement) were retained, while SPAM, T1, and T2 noise sources were removed. Graphs were run with 2-regular cyclic topologies from 5 to 10 qubits. Noise was applied in three noise levels (BASELINE, TARGET, DESIRED) to allow evaluation of different error magnitudes, with all parameters configurable via CSV input.

4.2.4. Results

All experiment data were collected automatically and stored in CSV format at the end of each execution. The following variables were recorded:

- **timestamp**: execution time and date.
- **qpu_emulator**: emulator used (MPS or LinAlg).
- **emulator_params**: parameters defining emulator configuration (e.g., target fidelity).
- **gpu**: whether GPU acceleration was used.
- **optimization_method**: optimizer employed (COBYLA or Nelder-Mead).
- **optimization_params**: configuration of the optimizer (initial parameters, tolerance, iteration limit).
- **computational_time**: total runtime of the execution.
- **num_qubits**: number of qubits in the circuit.
- **energy**: objective function value obtained from the optimizer.
- **job_type**: type of quantum algorithm (QAOA).
- **job_params**: additional parameters (e.g., ansatz depth).

- **circuit_optimal_params**: optimized variational parameters returned by the emulator.
- **sampling_nbshots**: number of shots used for circuit sampling.
- **hamiltonian**: problem Hamiltonian used.
- **graph_num_nodes**, **graph_num_edges**: structural properties of the benchmark graph.
- **graph_edges**: list of graph edges.
- **partition_set_0**, **partition_set_1**: resulting MaxCut partitions.
- **fidelity**: calculated using Qaptiva's `compute_fidelity` function to quantify the effect of the Depolarizing Channel.

Execution times with only the Depolarizing Channel remained higher than noiseless QAOA, but energy values were very similar to the LinAlg QPU. The three noise layers showed comparable execution times, and energy errors were slightly variable but without a clear trend.

4.2.5. Analysis

The collected data were analyzed primarily in terms of:

- **Fidelity**: measured as the truncation fidelity product for MPS emulations.
- **Computational time**: total execution time for each emulator and combination of parameters.
- **Energy**: value representing the expected cost function for MaxCut.

Visualization was performed using line plots and bar charts to compare emulator and optimizer performance across graph sizes. All plots are presented in Chapter 5.

From the analysis of the raw CSV data, several conclusions were drawn:

- The Nelder–Mead optimizer was found to be unsuitable for these experiments because of its excessive runtime. As a result, the COBYLA optimizer was selected as the preferred approach.
- The MPS emulator exhibited unexpectedly long execution times compared to the LinAlg emulator, suggesting potential inefficiencies or misconfiguration.
- Execution with only the Depolarizing Channel simplified the benchmark while still capturing realistic energy deviations.
- `eps_1` and `eps_2` allowed controlled tuning of gate errors for single- and two-qubit operations.
- Energy values were largely unaffected by removing SPAM and T1/T2 noise.
- Fidelity computation provided complementary information on the effect of noise on the quantum state.

These findings motivated a refined second approach with expanded parameterization, improved data extraction, and further emulator calibration.

4.3. Second Iteration

This section describes the second phase of the project, which built on what was learned from the first iteration. The goal was to extend the study to new approaches, clarify some of the issues previously observed, and better understand how the emulators behave under different conditions.

4.3.1. Research

In this phase, we expanded the study to include the Recursive Quantum Approximate Optimization Algorithm. This algorithm extends QAOA by reducing the problem recursively, which has a demonstrated analytical advantage over the original QAOA in complete graphs.

We also looked more closely at the problems found in the first iteration, especially the effect of node ordering when using a linear topology in the MPS emulator. Through both review and testing, we confirmed that the way qubits are connected and ordered has a direct impact on circuit construction. Nonlocal interactions force the insertion of additional *SWAP* gates, which make MPS emulations slower and more memory-demanding.

Finally, we broadened the theoretical scope to explore how graph density influences emulator performance. The aim was to see how increasing connectivity affects runtime and fidelity across emulators.

The main questions guiding this iteration were:

- How does RQAOA compare to QAOA under similar emulation conditions (not only complete graphs)?
- How do graph density and topology influence runtime and fidelity, especially for MPS?

4.3.2. Implementation

The implementation was extended from the previous version to support larger and more varied configurations:

- The number of qubits was increased up to 30.
- Regular graphs with different degrees were generated, from sparse (2-regular) to dense (20-regular), to study how connectivity affects performance.
- The ansatz depth was tested with values of 1, 3, and 5.
- Two random seeds (14 and 34) were used to initialize the optimizer, to check if this affected the final energy.

Noise models were also added to both QAOA and RQAOA circuits to bring emulations closer to real hardware behavior.

Implementing RQAOA required some additional work, since each recursive step removes nodes from the problem. This meant that we had to keep track of the "erased" nodes and the corresponding reduced Hamiltonians in order to reconstruct the solution of the original problem afterwards. Because qubit ordering matters in the Qaptiva environment, the qubit indices had to be updated carefully at every recursion step.

During development, we also encountered an issue where non-unitary matrices were being applied to MPS circuits, which affected the normalization of the quantum state. This was traced to a bug in the emulator and was fixed after updating to a newer Qaptiva version.

4.3.3. Execution

For this iteration, we limited the experiments to the *COBYLA* optimizer, since it had proven to be the fastest in the previous phase. The focus here was not to find the best possible energy but to observe performance trends and emulator behavior.

Experiments were carried out over the full range of parameters described above, combining different graph densities, ansatz depths, and random seeds. Even with these adjustments, the MPS emulator still showed long runtimes—up to around 7 hours for a 20-node circuit—with a truncation fidelity around 0.8.

Additionally, the code from the first iteration was formalized and cleaned up. Only the Depolarizing Channel noise was used, due to the large execution times and the negligible differences in energy values. Gate times for CNOT, H, RX, PH, state preparation, and measurement were retained, while all other noise sources (SPAM, T1/T2) were removed. The Depolarizing Channel is characterized by two parameters: `eps_1` and `eps_2`, representing the probability of error for single-qubit and two-qubit gates, respectively. Graphs were expanded compared to the first iteration, including 2-regular and 4-regular graphs from 5 to 12 qubits, as far as execution time allowed. Noise was applied in three layers (BASELINE, TARGET, DESIRED), with slightly increasing intensity, based [84]. These configurations are read from a CSV file to facilitate easy benchmark updates.

4.3.4. Results

In addition to the variables recorded in the first iteration, this phase included new metrics to give a more complete picture of the execution. The new fields were:

- **emulator_device**: hardware or emulator identifier (e.g., `Qaptiva 800s`).

- **seed**: random seed used for optimizer initialization.
- **total_process_computational_time**: total time of the whole execution process.
- **optimization_time**: time spent during optimization.
- **preprocessing_sample_job_time**: time used to prepare and sample the circuit.
- **run_sample_time**: time for the circuit run and sampling step.
- **best_energy_found**: final sampled energy of the best solution.
- **best_number_edges_cut_found**: number of edges in the best MaxCut solution.
- **truncation_fidelity_product**: overall fidelity from MPS truncations.
- **depth_circuit**: circuit depth.
- **size_circuit**: total number of gates.
- **optimization_best_energy**: energy value from the optimizer before sampling.
- **fidelity**: computed using Qaptiva's `compute_fidelity` function to measure the overlap between ideal and noisy states.

Execution times remained several orders of magnitude higher than noiseless QAOA, while energy values stayed similar to the LinAlg QPU. Among the three noise layers, execution times were comparable, and energy errors showed slight variations without a clear trend.

4.3.5. Analysis

The analysis focused on how graph density, ansatz depth, and initial seed affected the performance and quality of results.

The main observations were as follows:

- Increasing the graph degree and size quickly raised computation time for the MPS emulator, showing its sensitivity to connectivity. This result can be observed in the different time scales of Figure 5.2 and Figure 5.6 in Chapter 5.
- Higher ansatz depths gave slightly better energy convergence but increased the runtime considerably.
- Different random seeds produced small variations in the final energies with no clear pattern; therefore, we retained a single seed in subsequent iterations to reduce emulation time.
- RQAOA turned out to be computationally more expensive than standard QAOA, and the empirical results did not show clear benefits that justified the additional cost. This conclusion was drawn from Figures 5.2 and 5.1.
- The MPS emulator still required very long execution times, especially for dense graphs, suggesting that this method may not be the best fit for QAOA-based experiments.
- Keeping only the Depolarizing Channel substantially reduced computational complexity compared to the full noise model.
- Gate times for all operations were retained to reflect realistic hardware timing.
- `eps_1` and `eps_2` provided an interpretable way to tune the depolarizing error strength for single- and two-qubit gates, respectively.
- Energy values were largely unaffected by removing SPAM and T1/T2 noise, confirming that the Depolarizing Channel captures the dominant error behavior for these small graph instances.
- Fidelity computation allowed direct quantification of the effect of noise on the quantum state, providing a complementary perspective to energy-based evaluation.

Overall, the second iteration helped clarify how emulator configurations, circuit depth, and graph structure affect performance, while streamlining noisy emulations for practical benchmarking.

4.4. Third Iteration

This third phase of the work was focused on identifying circuit structures that better exploit the strengths of the MPS emulator, while still allowing for a controlled increase in emulation complexity. The main motivation was to understand under which conditions MPS emulation provides computational advantages, and to complement this with an analysis of entanglement in the generated circuits.

4.4.1. Research

After observing in the previous iterations that QAOA and RQAOA circuits were not ideal for MPS-based emulations—mainly due to their connectivity and gate layout—we decided to explore an alternative class of circuits more suitable for tensor network representations. Following this line, we investigated the *Cheng-generated random circuits*, as described in Section 2. These circuits are generated randomly by fixing the number of qubits and circuit depth (i.e., number of layers), allowing a progressive control over complexity and entanglement.

This setup provided a convenient way to stress-test both emulators under gradually increasing workloads. In parallel, we extended the theoretical study to include measures of quantum entanglement, as these are directly linked to MPS performance. High entanglement is known to increase the required bond dimension and thus the memory and time costs of tensor network emulations. Consequently, we focused on the *Von Neumann Entropy* as the main entanglement measure to characterize the circuits.

The main research objectives of this iteration were:

- To identify the regions where MPS outperforms LinAlg, and vice versa.
- To analyze how entanglement evolves with increasing circuit depth and how it correlates with emulation fidelity and runtime.
- To characterize the computational limits of both emulators under controlled circuit configurations.

4.4.2. Implementation

For the implementation, we created a dataset of pre-generated random circuits to ensure full reproducibility across emulators. Each circuit was defined by two parameters: number of qubits and number of layers. We generated circuits with `num_qubits` ranging from 5 to 60 and `layers` ranging from 5 to 75, resulting in a diverse set of configurations covering both shallow and deep regimes.

We also decided to try another approach for the MPS emulator. Instead of targeting a fixed fidelity threshold, we progressively increased the maximum allowed bond dimension, observing how the fidelity evolved with this parameter. The process was halted once the achieved fidelity reached 0.95, which allowed us to study the tradeoff between accuracy, entanglement, and computational cost.

To quantify entanglement, we used the *Von Neumann Entropy* across all tensor sites. For MPS, this metric was directly obtained through Qaptiva's built-in functionality, while for LinAlg we implemented an equivalent computation manually to ensure consistency between both emulators.

4.4.3. Execution

The execution phase consisted of running the complete set of pre-generated circuits on both emulators. For each configuration (combination of number of qubits and layers), we recorded runtime, fidelity, and entanglement metrics.

Because of the computational load, the experiments were run in several sessions, with automatic execution and results saved to CSV files as done in previous iterations. Each circuit was executed multiple times with gradually increasing bond dimension (for MPS) until the fidelity reached 0.95. For LinAlg, the executions were limited by available memory, so we stopped the tests at around 35 qubits.

All experiments were conducted without noise models, as the goal was to isolate emulator-related performance differences. We also included a count of two-qubit gates in each circuit to correlate gate density with the observed runtime and fidelity decreased, since entanglement growth is typically driven by two-qubit operations and MPS need for more steps to perform this type of gates.

4.4.4. Results

In addition to the parameters already recorded in previous iterations, this iteration included new fields specifically included to capture answer the questions for this iteration characteristics:

- **bond_dimension**: maximum bond dimension reached during MPS emulation.
- **von_neumann_entropy_array**: list of Von Neumann entropies per tensor site.
- **num_two_qubit_gates**: total number of two-qubit gates in the circuit.

These additions provided the means to study how circuit structure and entanglement growth influence emulator scalability and efficiency.

4.4.5. Analysis

The analysis in this iteration focused on identifying the boundary conditions under which MPS performs better than LinAlg in terms of both computational time and scalability.

From the experimental data, we observed the following trends:

- MPS could emulate a significantly larger number of qubits compared to LinAlg, which typically reached its limit around 35 qubits due to exponential memory requirements (Figures 5.18 and 5.19).
- MPS performance deteriorated with increasing circuit depth, confirming that deep circuits induce higher entanglement and larger bond dimensions, which in turn slow down the emulation.
- LinAlg scaled well with circuit depth as long as the qubit number remained moderate.
- A tradeoff was observed between circuit size and runtime: for lower qubit counts, LinAlg remained faster, but as the number of qubits increased, MPS became more efficient despite its truncation overhead.

Overall, this iteration helped clarify how both emulators behave under different conditions. MPS showed better scalability with larger numbers of qubits, but its performance decreased for deeper and more entangled circuits. On the other hand, LinAlg handled shallower or medium-sized circuits more efficiently. We present in Table 4.2 a summary of the methodology iterations.

In the next section, we will continue with a detailed presentation of the results, including the exact plots, quantitative comparisons, and further discussion of the observed trends.

Table 4.2: Summary of followed methodology across the three iterations descripted above.

Phase	First Iteration	Second Iteration	Third Iteration
Research	Investigated QAOA behavior on 2-regular cyclic graphs; compared LinAlg and MPS emulators; studied noise effects (Depolarizing, Idle, SPAM). Hypotheses on emulator scaling and performance.	Expanded to RQAOA; studied effect of graph density, node ordering, and topology; clarified issues from first iteration; analyzed impact of problem graph connectivity on runtime and fidelity.	Explored circuits optimized for MPS (Cheng-generated random circuits); focused on entanglement evolution (Von Neumann entropy); identified conditions where MPS outperforms LinAlg.
Implementation	Developed agnostic class for LinAlg and MPS; included noise modeling using Qaptiva NoisyQProc; configured BASELINE, Target, Desired noise levels; CSV-driven reproducible setup.	Extended to larger qubit numbers (up to 30), varied graph densities (2-20 regular), multiple ansatz depths, and multiple seeds; handled RQAOA recursive reduction; applied Depolarizing noise only.	Generated dataset of random circuits (5-60 qubits, 5-75 layers); MPS bond dimension progressively increased to target fidelity 0.95; tracked entanglement metrics; ensured reproducibility.
Execution	Ran QAOA circuits (5-11 qubits) using Nelder-Mead and COBYLA; MPS dynamic fidelity-based truncation; custom linear topology for MPS; tested noise impact with Depolarizing Channel.	Used COBYLA optimizer only; executed circuits across graph densities, ansatz depths, and seeds; tracked execution times, energy, and fidelity; simplified noise to Depolarizing Channel.	Executed full set of pre-generated circuits on both emulators; recorded runtime, fidelity, entanglement, and two-qubit gate counts; progressive bond dimension increase for MPS; no noise applied.
Results	Collected CSV data: energy, fidelity, computational time, emulator, graph properties, optimizer info; simplified noise analysis.	Added metrics: emulator device, seed, total process time, optimization time, circuit sampling times, best energy, edges cut, truncation fidelity, circuit depth/size; observed runtime and fidelity trends.	Added metrics: bond dimension, Von Neumann entropy array, number of two-qubit gates; analyzed effect of entanglement and circuit depth on emulator scalability.
Analysis	Compared emulators and optimizers; observed MPS runtime inefficiencies; evaluated impact of Depolarizing noise; energy largely unaffected by SPAM/T1/T2 removal; fidelity used to quantify noise effects.	Analyzed graph density, ansatz depth, and seed effects; observed MPS sensitivity to connectivity; RQAOA more expensive; Depolarizing Channel sufficient to capture noise behavior; retained gate times.	Evaluated performance boundaries; MPS scalable for larger qubits but slower with deeper/highly entangled circuits; LinAlg efficient for shallow/medium circuits; tradeoff between qubit number and runtime identified; fidelity tracked entanglement effects.

5

Results

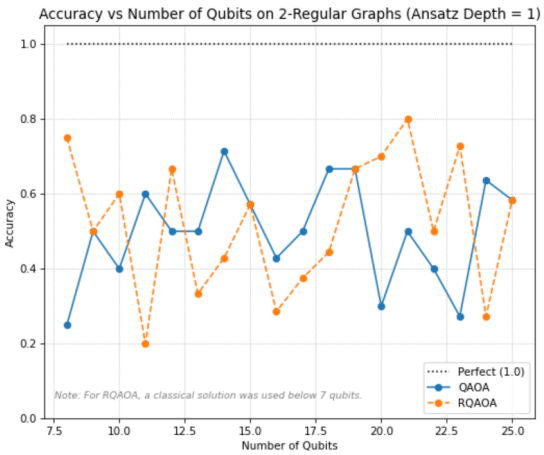
In this section, we take a closer look at the results that were briefly introduced in Section 4. The goal is to interpret these findings with respect to the research questions presented in Section 1. The analysis focuses on the performance of QAOA and RQAOA when applied to MaxCut problem instances on 2-regular, 4-regular, and complete graphs. The main key performance indicators (KPIs) considered are execution time and accuracy. It is important to note that the reported accuracy values do not necessarily correspond to the optimal solutions, as reaching the global optimum is beyond the scope of this work. Instead, accuracy comparisons are performed under identical optimization conditions, which may not be optimal for either algorithm. In addition, we will be spliting results with respect to their corresponding ansatz depth for the research questions where QAOA is involved. The section is organized around the main research objectives, addressing each of them in turn to provide clear answers, discuss the reasoning behind the observed results, and highlight the main strengths, limitations, and implications of the emulator performance within the context of this study.

5.1. Performance comparison between QAOA and RQAOA in a statevector emulator.

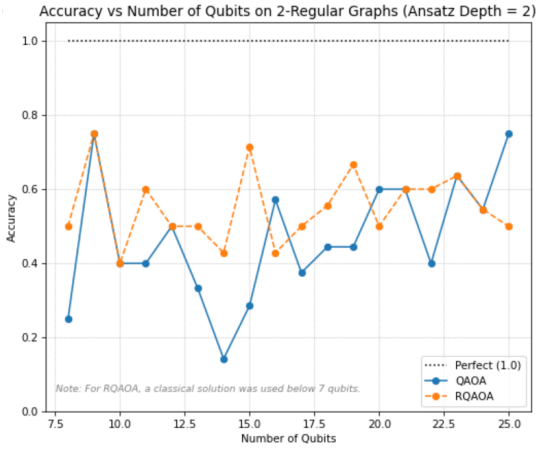
This subsection addresses the first research question, which examines how the performance of QAOA compares with RQAOA across the considered graph topologies. Under the same optimization settings, we analyze how both algorithms perform in terms of accuracy and execution time. The goal is to verify whether the obtained results are consistent with those reported in [27], where RQAOA was shown to outperform QAOA for complete graphs, and to assess whether this trend also holds for other graph structures such as 2-regular and 4-regular instances. For all experiments we used the *COBYLA* optimizer from *Scipy* Python package with a *maxiter=200*. For all instances, the classical limit of RQAOA—defined as the number of remaining nodes for which a brute-force solution is applied—was set to 7.

5.1.1. 2-Regular Graphs

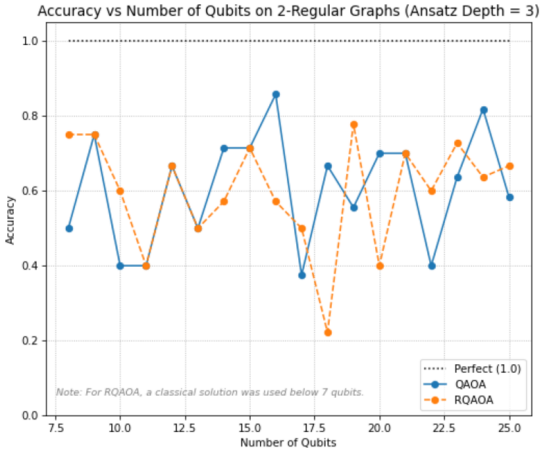
Here we present the results for the 2-regular graphs. Figure 5.1 shows how the accuracy of both algorithms evolves with ansatz depth. In each plot, the accuracy is measured relative to the optimal solution, obtained via a brute-force algorithm and indicated by the black dashed line at 1.0; this reference is maintained throughout the results section. The tested graphs range from 7 to 25 nodes, corresponding directly to the number of qubits. For the 2-regular graphs, no clear trend emerges across different ansatz depths. While RQAOA performs slightly better than QAOA in most of the instances of Figure 5.1b, this improvement is not consistent for all cases.



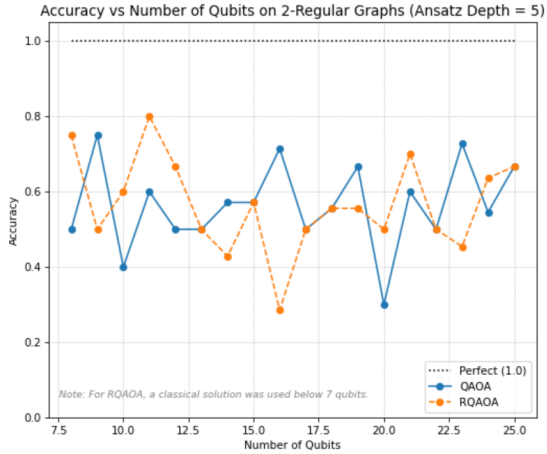
(a) Ansatz Depth 1



(b) Ansatz Depth 2



(c) Ansatz Depth 3

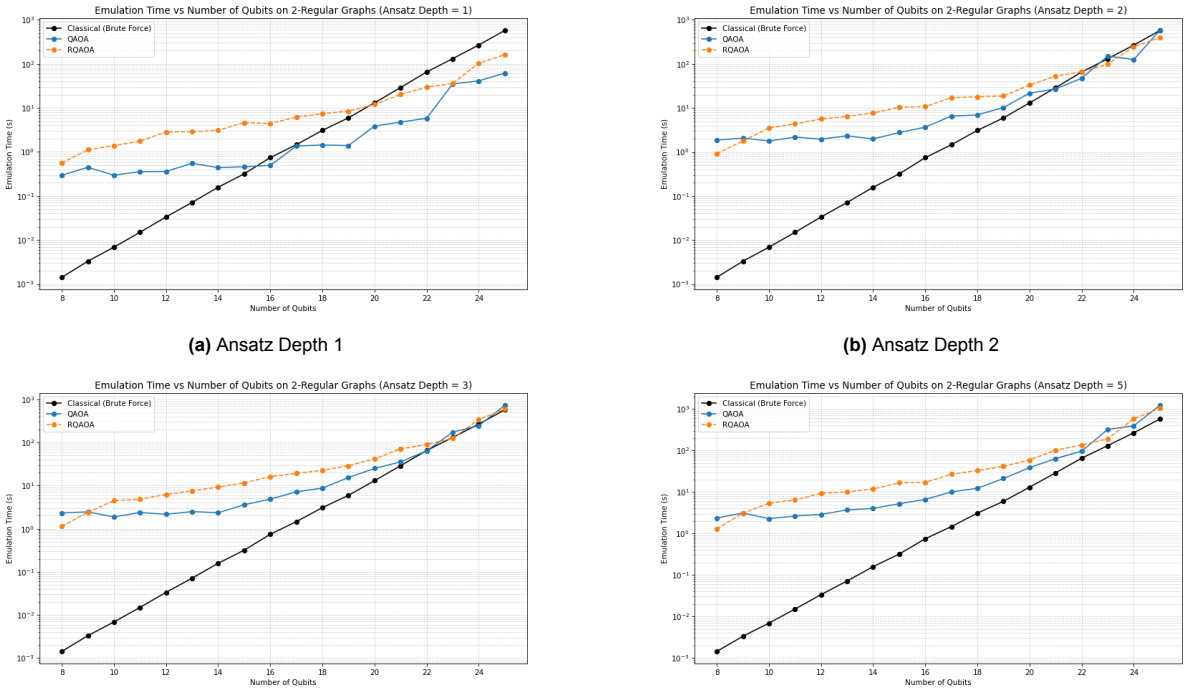


(d) Ansatz Depth 5

Figure 5.1: Accuracy comparison for different ansatz depths. Each subfigure corresponds to a specific depth.

Regarding computational time, RQAOA generally requires more resources than QAOA as shown in Figure 5.2, which is expected given that it performs multiple optimization steps based on QAOA. Interestingly, as the problem size increases, this time gap decreases, and for larger ansatz depths, RQAOA can even become faster than standard QAOA. This suggests that for even larger problem instances, RQAOA could offer comparable accuracy with improved time efficiency.

It is also important to note that both algorithms are compared against a classical brute-force method, which achieves better accuracy and shorter runtimes for these small instances. With the exception of Ansatz Depth 1, the brute-force approach remains faster across all tested instances. However, while the initial runtime gap between the classical and quantum-based methods is significant, it steadily decreases as the problem size increases, eventually leading to comparable runtimes. This closing gap suggests more favorable scaling behavior for QAOA and RQAOA, even though they do not outperform the brute-force method within the explored problem sizes.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

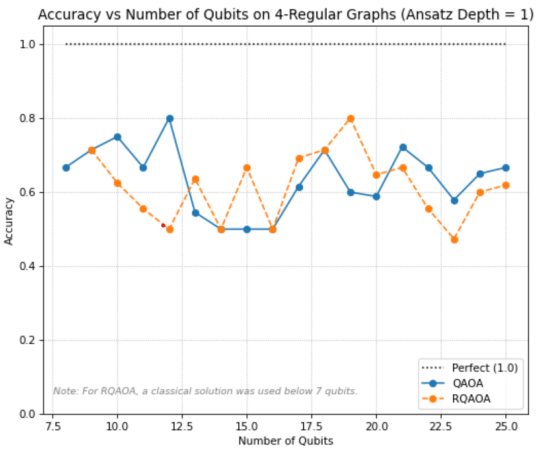
(c) Ansatz Depth 3

(d) Ansatz Depth 5

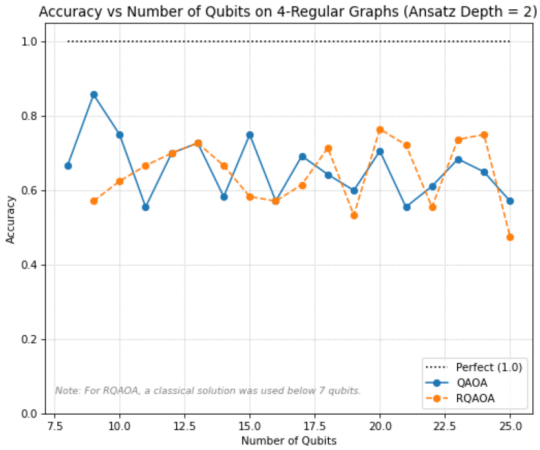
Figure 5.2: Time comparison for different ansatz depths. Each subfigure corresponds to a specific depth.

5.1.2. 4-Regular Graphs

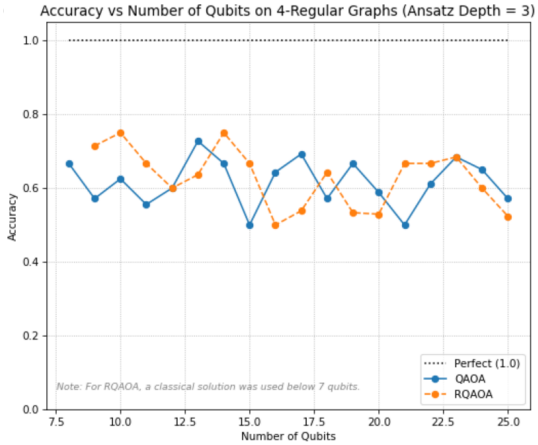
We now proceed to evaluate both methods on 4-regular graphs. As shown in Figure 5.3, the overall accuracy has improved for both algorithms compared to the 2-regular case, and the results are more consistent. This contrasts with the earlier instances, where accuracies were more variable. Despite these improvements, no method can be identified as a clear winner. RQAOA generally produces more robust solutions with fewer fluctuations, whereas QAOA performs better in several instances at Ansatz Depth 5, as illustrated in Subfigure 5.3d.



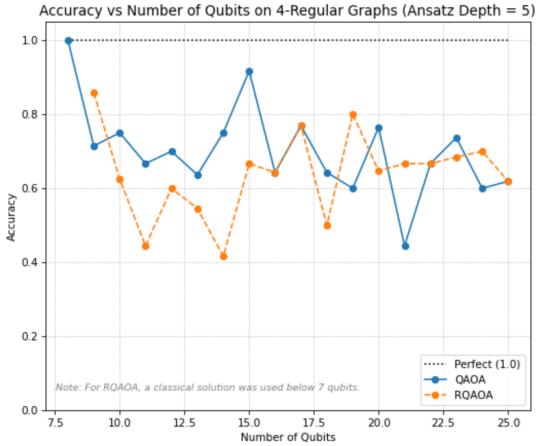
(a) Ansatz Depth 1



(b) Ansatz Depth 2



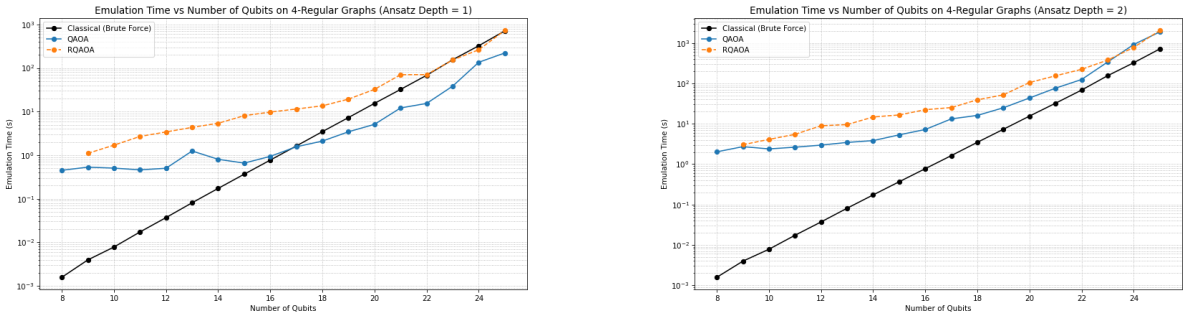
(c) Ansatz Depth 3



(d) Ansatz Depth 5

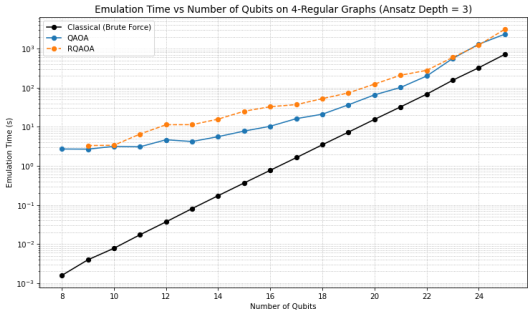
Figure 5.3: Accuracy comparison for different ansatz depths on 4-regular graphs. Each subfigure corresponds to a specific depth.

Regarding computational time, similar trends to the 2-regular case are observed: RQAOA generally requires more computational resources but exhibits better scaling compared to QAOA. A key difference in this scenario is that both methods are slower than the reference brute-force approach, except for the case where the Anstaz Depth is equal to 1. Although a reduction in the runtime gap is still observed for larger problem sizes at increased ansatz depths, the performances do not converge to comparable values. Moreover, as the ansatz depth increases, the remaining gap becomes more pronounced, indicating that higher circuit depths further disadvantage the quantum-based approaches for this class of instances within the explored regime.

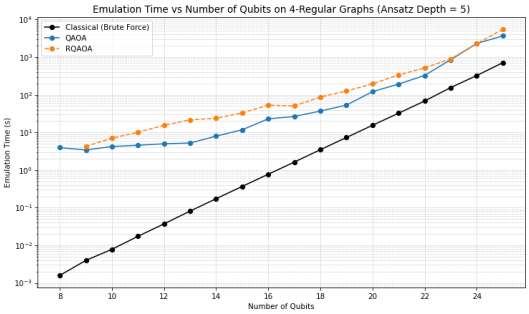


(a) Ansatz Depth 1

(b) Ansatz Depth 2



(c) Ansatz Depth 3

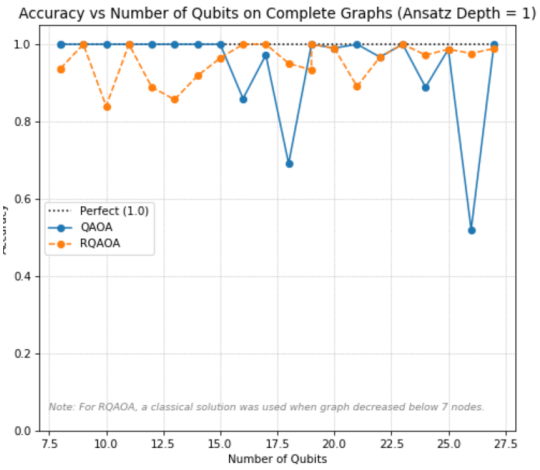


(d) Ansatz Depth 5

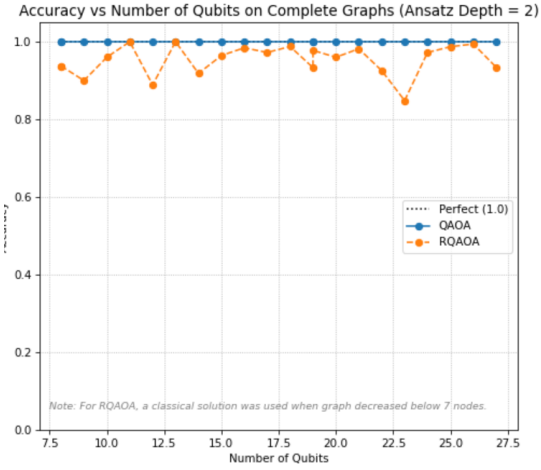
Figure 5.4: Execution time comparison for different ansatz depths on 4-regular graphs. Each subfigure corresponds to a specific depth.

5.1.3. Complete Graphs

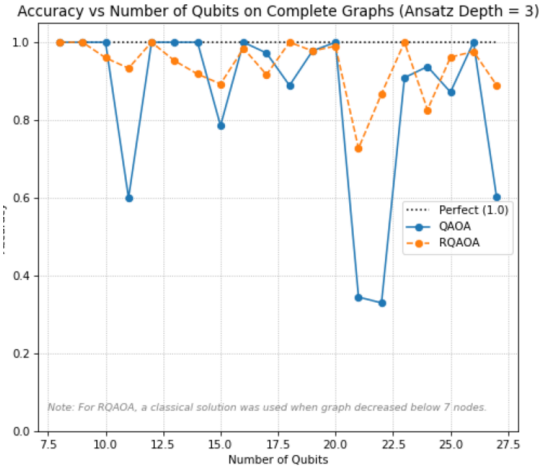
Finally, we evaluate both methods on complete graphs, extending the emulation to 27 nodes. Overall, the results continue to improve, indicating that the higher connectivity of these graphs helps the algorithms reach better solutions. Notably, at ansatz depth 2 (Subfigure 5.5b), QAOA achieves the optimal solution in all cases by getting an accuracy of 1 in all experimented instances. RQAOA, on the other hand, demonstrates remarkable robustness: while QAOA occasionally outperforms it, these successes are less consistent, as evidenced by multiple lower peaks that become more pronounced at higher ansatz depths, sometimes dropping to zero accuracy (see Subfigure 5.5d).



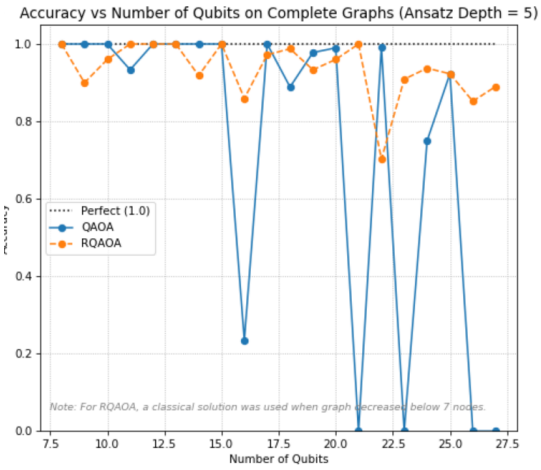
(a) Ansatz Depth 1



(b) Ansatz Depth 2



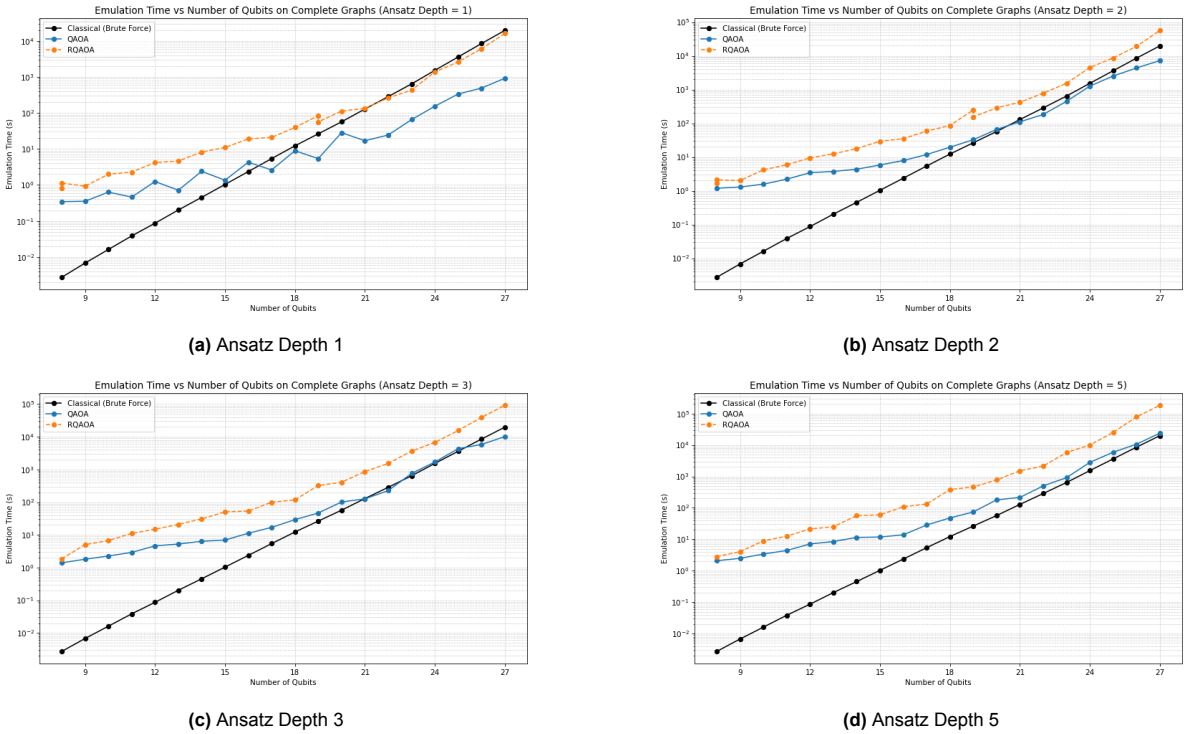
(c) Ansatz Depth 3



(d) Ansatz Depth 5

Figure 5.5: Accuracy comparison for different ansatz depths on complete graphs. Each subfigure corresponds to a specific depth.

Regarding execution time, the trends observed in previous cases persist. RQAOA remains more computationally expensive than QAOA, and both methods are outperformed by the brute-force algorithm. For these complete graph instances, runtimes can reach up to 10^5 seconds for practically all ansatz depths as shown in Figure 5.6, corresponding to approximately one day of emulation per case.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.6: Execution time comparison for different ansatz depths on complete graphs. Each subfigure corresponds to a specific depth.

5.2. Performance comparison between Noisy and Noiseless emulators with QAOA.

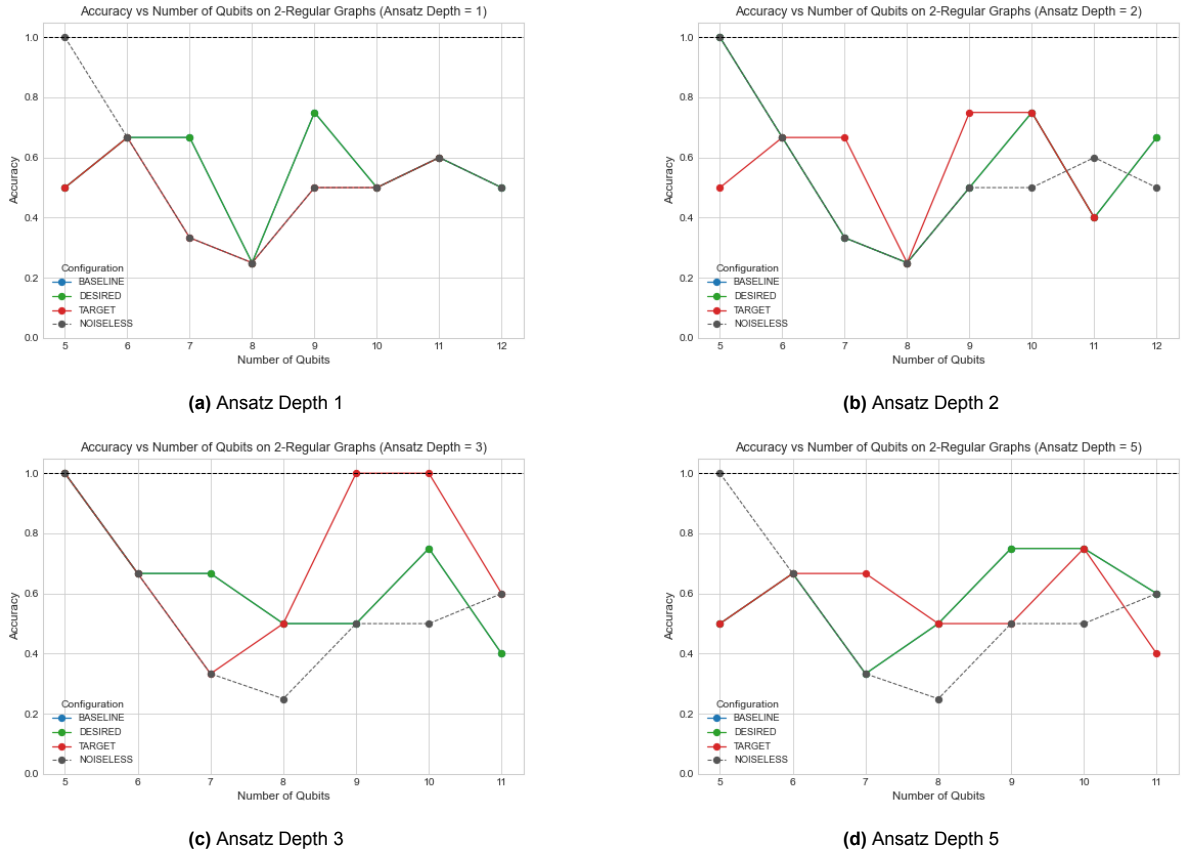
This section addresses the second research question, which investigates how noise affects the performance of QAOA across the different graph structures. Specifically, we compare the results obtained from noisy quantum emulations with those from a noiseless classical emulator that serves as a reference for the ideal behavior. The analysis focuses on both accuracy and computational time as done before, with the aim of quantifying the degradation introduced by noise.

We will contrast the BASELINE, DESIRED, and TARGET noisy configurations explained in Section 4 with the NOISELESS benchmark (corresponding to the `LinAlg` emulator). We aim to determine whether the observed trends in the already-presented results remain consistent across 2-regular and 4-regular graph instances for noisy scenarios.

5.2.1. 2-Regular Graphs

We now present the results obtained for the 2-regular graphs. Figure 5.7 illustrates how the accuracy evolves with the ansatz depth. The accuracy values are expressed relative to the optimal classical solution, represented by the black dashed line at 1.0, as in previous analyses. The tested graphs range from 5 to 12 nodes for ansatz depths of 1 and 2. Due to time constraints, the noisy emulations for ansatz depths of 3 and 5 were limited to instances up to 11 nodes.

The first observation is that the BASELINE and DESIRED configurations yield identical accuracy across all instances, indicating that the parameter differences between these setups do not lead to measurable changes in the QAOA optimization. Furthermore, the accuracy does not display a clear dependence on either the ansatz depth or the problem size. Although no configuration consistently outperforms the others, it is remarkable that the noisy emulations occasionally surpass the noiseless reference. This behavior suggests that the presence of noise may help smooth the optimization landscape, facilitating convergence to better approximate solutions.



(a) Ansatz Depth 1

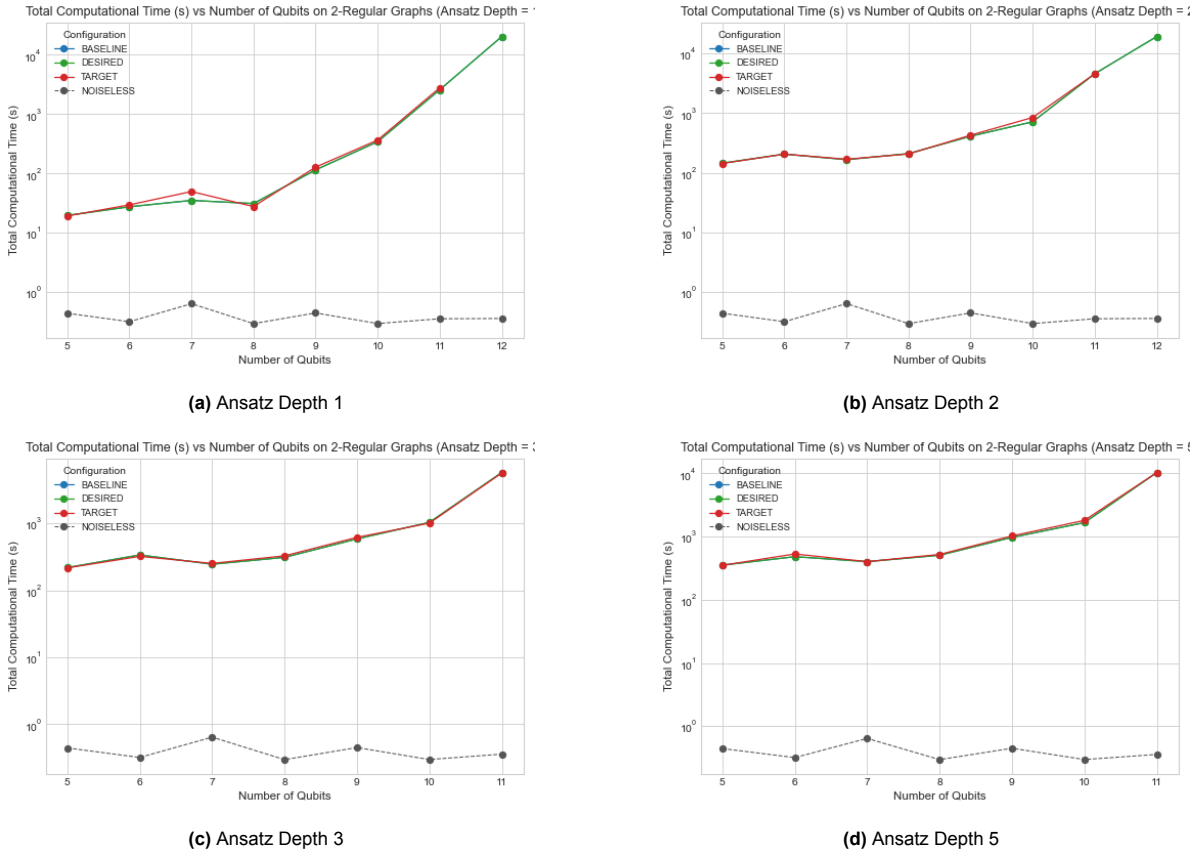
(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.7: Accuracy comparison for different ansatz depths on 2-regular graphs. Each subfigure corresponds to a specific ansatz depth, showing the performance of the BASELINE, DESIRED, TARGET, and NOISELESS configurations.

Regarding computational time, Fig. 5.8 shows that the emulation of noisy circuits requires substantially more time than the noiseless reference. Furthermore, the runtime increases approximately exponentially with system size, leading to a rapid growth in computational cost as the number of nodes increases. Interestingly, the computational times for all noise configurations (BASELINE, DESIRED, and TARGET) are nearly identical, and they do not vary significantly with the ansatz depth. This behavior can be explained by the fact that the emulation spends most of the time processing noise effects rather than executing the gates themselves. Consequently, under these noise conditions, the dominant factor affecting runtime is the size of the problem graph rather than the circuit depth.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

(c) Ansatz Depth 3

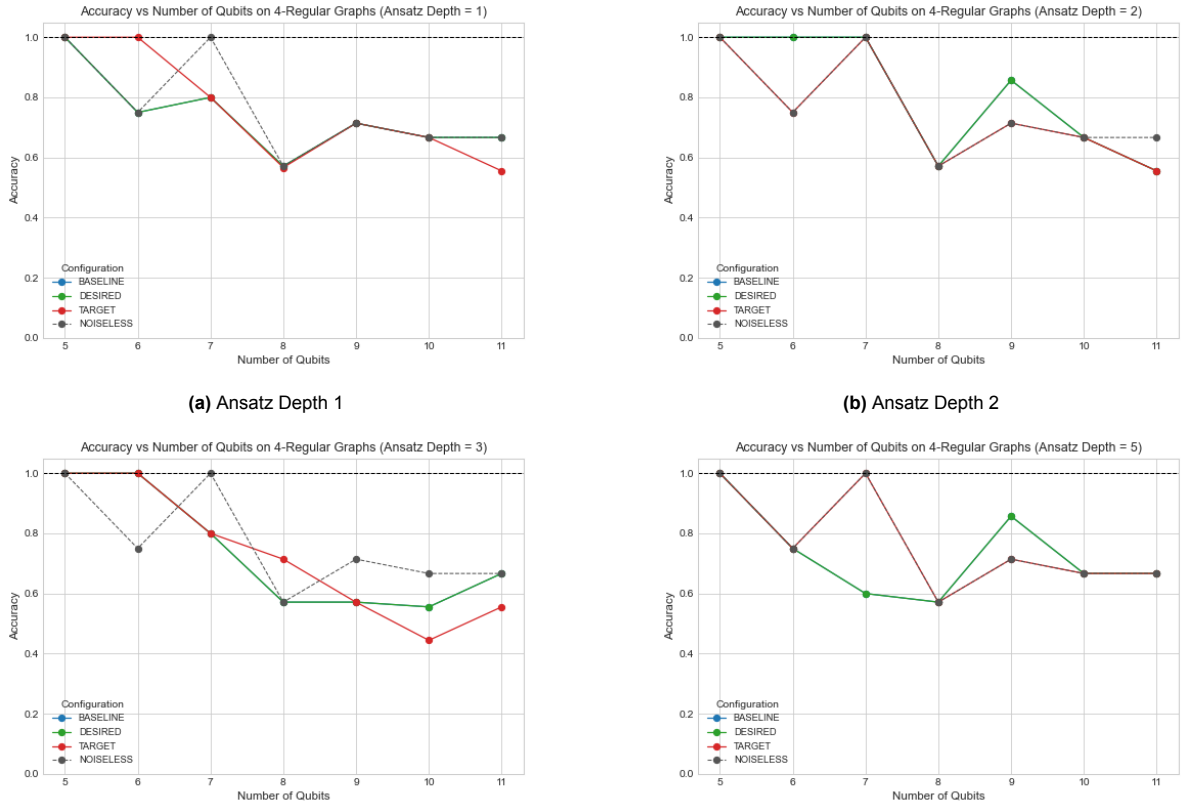
(d) Ansatz Depth 5

Figure 5.8: Total computational time for different ansatz depths on 2-regular graphs. The logarithmic scale highlights the exponential growth of runtime with system size.

5.2.2. 4-Regular Graphs

We now present the results obtained for the 4-regular graphs. Figure 5.9 illustrates how the accuracy evolves with the ansatz depth. The tested graphs range from 5 to 11 nodes for all cases.

The first observation is that accuracy has improved compared to the 2-regular graphs, reinforcing the idea that higher connectivity helps QAOA converge toward better solutions. However, in this case, the noiseless emulations achieve accuracy values close to those of the noisy emulations. This contrasts with the 2-regular case and suggests that the advantage provided by increased connectivity is partially mitigated by the presence of noise, which prevents the algorithm from fully leveraging the enhanced graph structure.



(a) Ansatz Depth 1

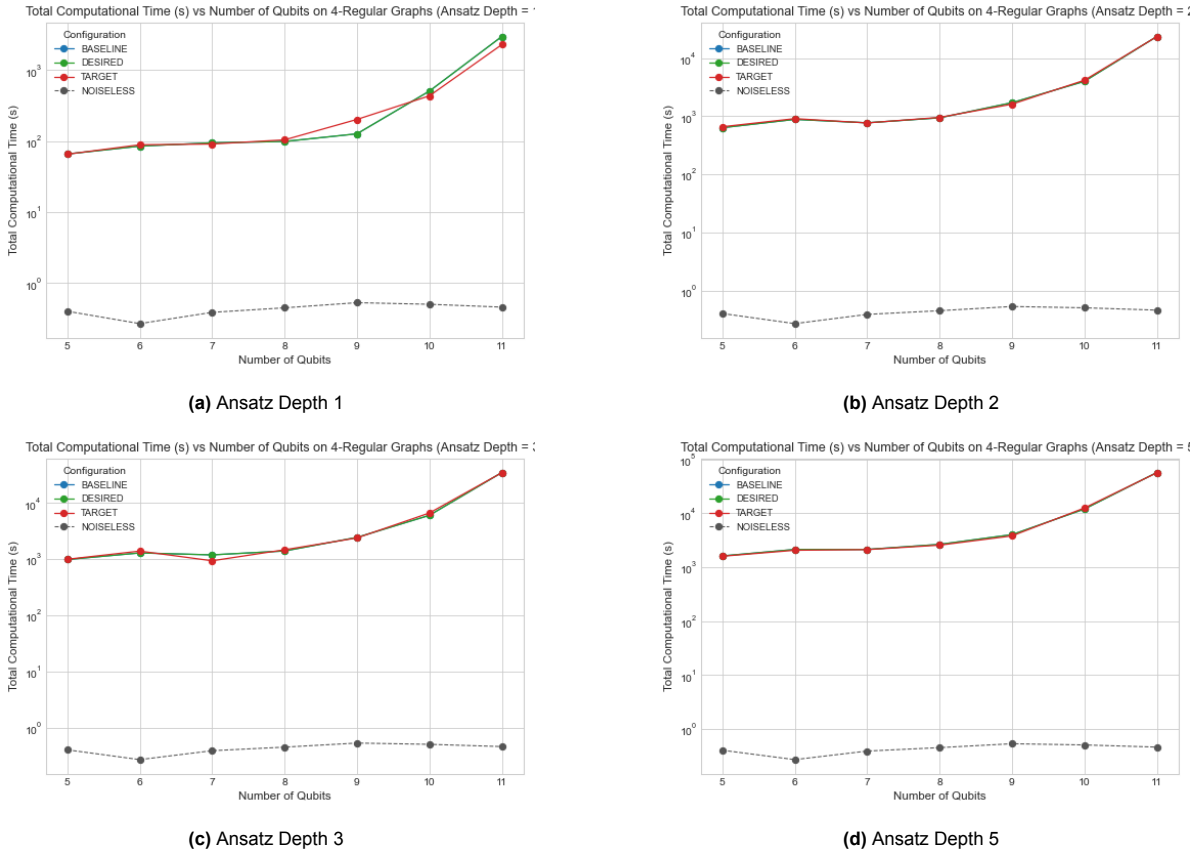
(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.9: Accuracy comparison for different ansatz depths on 4-regular graphs. Each subfigure corresponds to a specific ansatz depth, showing the performance of the BASELINE, DESIRED, TARGET, and NOISELESS configurations.

Regarding computational time, Figure 5.10 shows that the emulation of noisy circuits requires substantially more time compared to the noiseless reference, and the runtime scales poorly with system size. The computational times for all noise configurations (BASELINE, DESIRED, and TARGET) are nearly identical, with only minor fluctuations for the TARGET configuration. Although we observe a small increase in runtime with ansatz depth, this increase is smaller than expected given the corresponding rise in the number of gates, indicating that the dominant contribution to runtime comes from processing noise rather than gate execution. Most notably, the runtime has increased by nearly an order of magnitude compared to the 2-regular case, suggesting that for higher-connectivity graphs, introducing noise becomes computationally prohibitive. This is the main reason why experiments on the complete graph set have not been pursued.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.10: Total computational time for different ansatz depths on 4-regular graphs. The logarithmic scale highlights the exponential growth of runtime with system size.

5.3. Performance comparison between statevector and MPS emulators with QAOA.

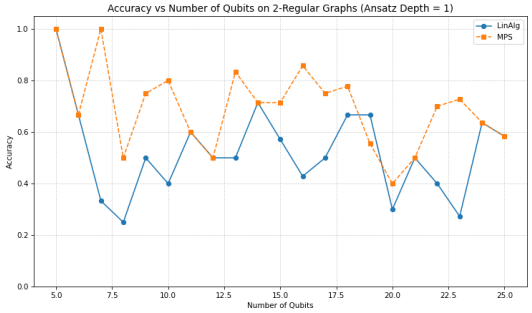
In this section, we analyze the performance of the two main emulation approaches—statevector and Matrix Product State—when emulating QAOA circuits. The focus is on understanding how their computational efficiency and accuracy evolve as the circuit depth increases. By comparing execution times and solution quality across the same problem instances used in the previous sections, we aim to identify regimes in which one emulator outperforms the other. For 2-regular and 4-regular graphs, emulations were performed on instances ranging from 5 to 25 nodes, while for complete graphs the comparison was limited to 20 nodes due to time-consuming constraints. Additionally, for similar reasons, RQAOA emulations were not performed using the MPS emulator, even though this functionality has been implemented.

For the MPS emulator, experiments were carried out using bond dimensions $\{2, 4, 8, 16, 32, 64\}$. The results presented in this section correspond to the configurations that achieved the highest accuracy among these bond dimensions. This approach enables a fair comparison with the statevector emulator under its best-performing settings. In cases where multiple bond dimensions yielded the same accuracy, the configuration with the smallest bond dimension was selected, as it provides comparable results with lower computational cost and faster emulation times.

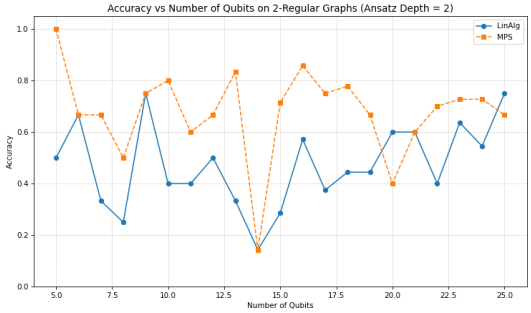
Although other parameter choices were possible, we selected this set of bond dimensions because it provides a wide range of potential fidelities — from typically low-fidelity emulations at bond dimension 2 to nearly exact emulations (up to around 10 qubits) when using a bond dimension of 64. For larger problem sizes, however, these values remain intentionally limited: higher bond dimensions would substantially increase both the number of required runs and the runtime of each emulation, rendering the experiments computationally infeasible within reasonable time constraints.

5.3.1. 2-Regular Graphs

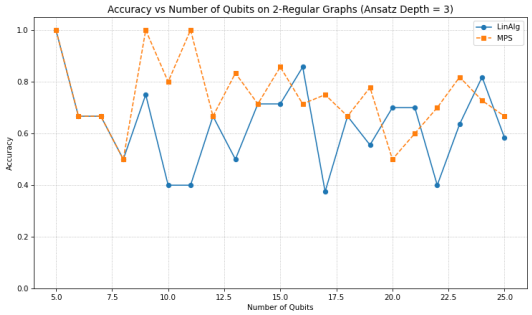
For the 2-regular graph instances, a clear advantage can be observed in favor of the *MPS* emulator, which consistently outperforms the *LinAlg* approach across nearly all instances and ansatz depths, as shown in Figure 5.11. Interestingly, increasing the ansatz depth does not lead to a noticeable improvement in accuracy. This behavior may be attributed to the relatively simple structure of 2-regular graphs, where the optimization landscape is already optimized at shallow circuit depths, making additional layers redundant and potentially even introducing noise in the parameter optimization process. An additional factor may be the choice of optimizer, which might not be optimal for this type of graph and could limit the effectiveness of deeper ansatz circuits.



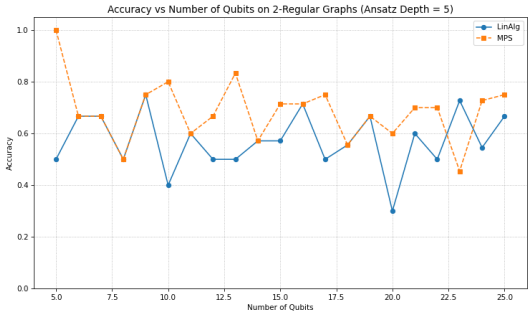
(a) Ansatz Depth 1



(b) Ansatz Depth 2



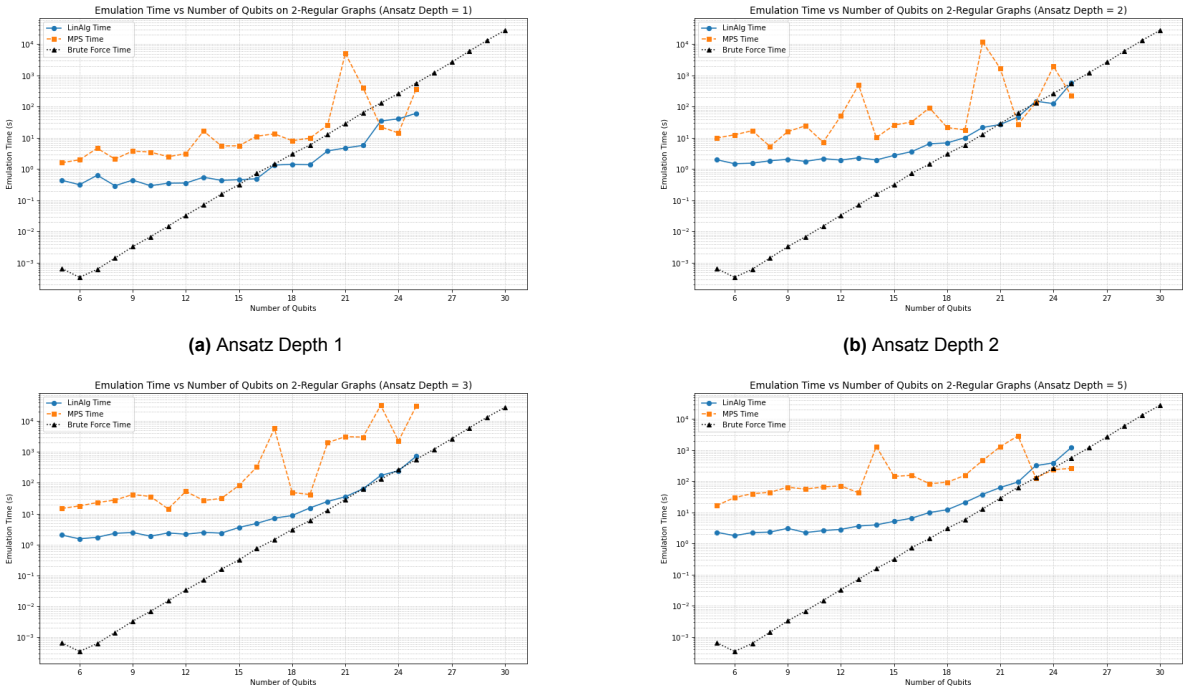
(c) Ansatz Depth 3



(d) Ansatz Depth 5

Figure 5.11: Accuracy comparison for different ansatz depths on 2-regular graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

Regarding computational time, although the MPS emulator generally achieves better accuracy, it is also computationally more demanding. The presence of pronounced time peaks can be attributed to the need for higher bond dimensions in certain instances. Combined with the structure of the QAOA ansatz—which involves multiple entangling gates—this makes the emulation particularly expensive, sometimes with a difference of 2 to 3 orders of magnitude. Nevertheless, as both the ansatz depth and graph size increase, this performance gap gradually narrows, and the MPS emulator even becomes more time-efficient in some cases, as observed in Subfigure 5.12d.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

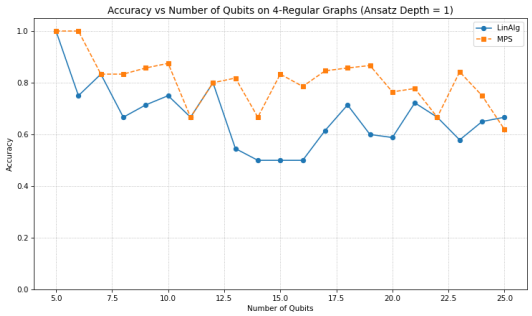
(c) Ansatz Depth 3

(d) Ansatz Depth 5

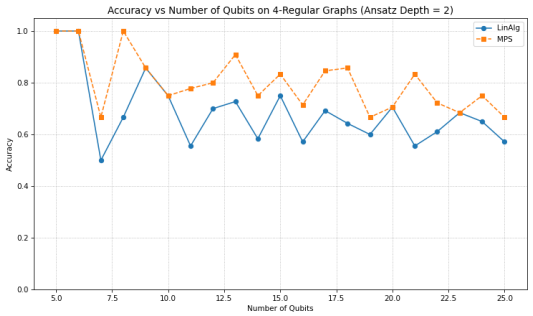
Figure 5.12: Execution time comparison for different ansatz depths on 2-regular graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

5.3.2. 4-Regular Graphs

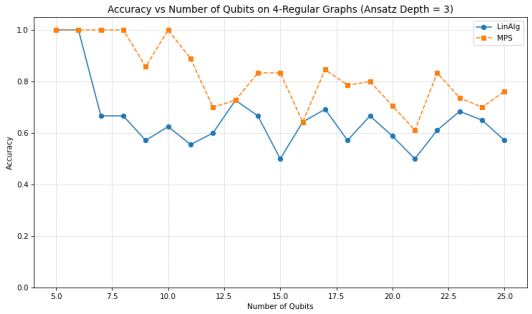
It is interesting to note that a similar trend to the previous section emerges here. The overall accuracy improves compared to the 2-regular graph instances, suggesting that the higher connectivity of 4-regular graphs aids convergence toward better solutions. The MPS emulator continues to outperform the statevector approach across all cases, as illustrated in Figure 5.13. However, its advantage slightly diminishes for the ansatz depth 5 scenario, where it achieves lower accuracy in a few instances—although it still maintains superior performance in most cases. Again, deeper circuits do not lead to better performance.



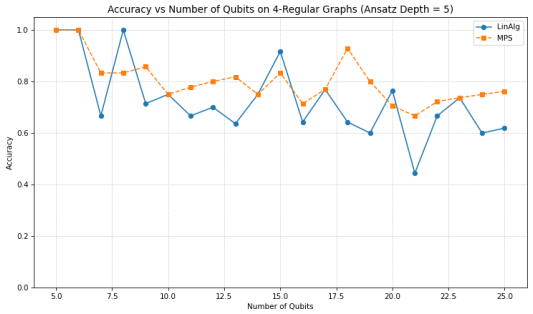
(a) Ansatz Depth 1



(b) Ansatz Depth 2



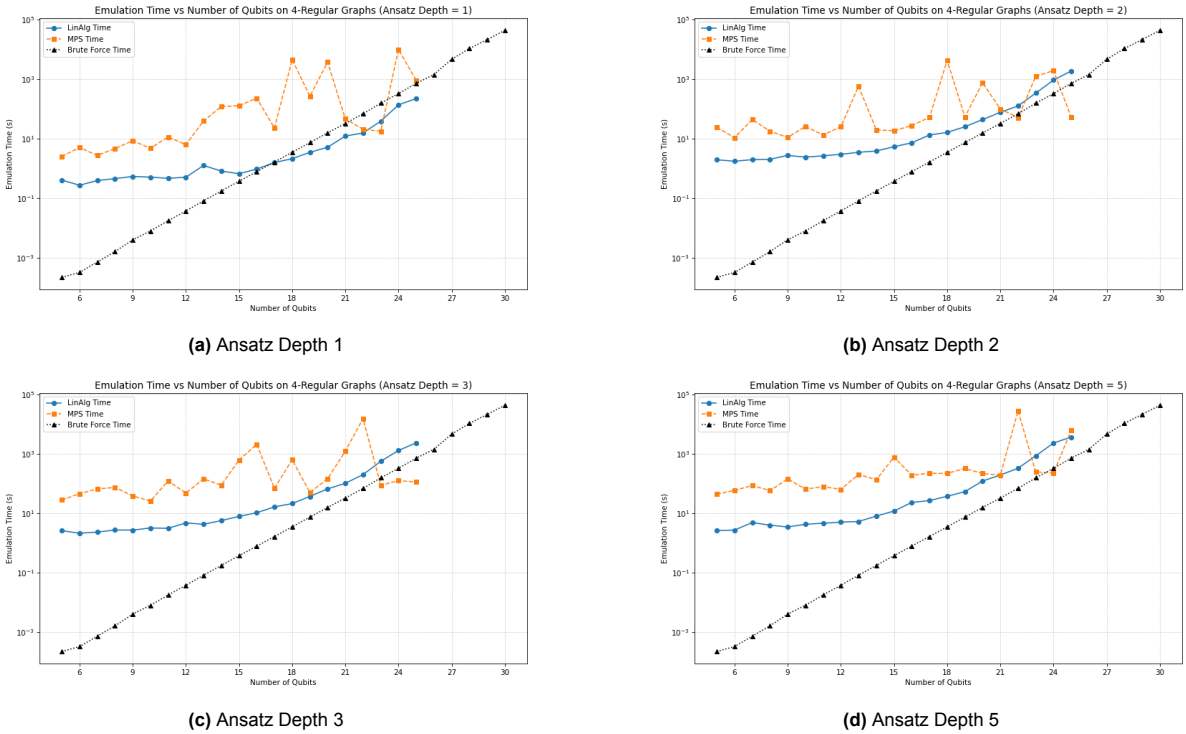
(c) Ansatz Depth 3



(d) Ansatz Depth 5

Figure 5.13: Accuracy comparison for different ansatz depths on 4-regular graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

Regarding execution time, we observe trends consistent with the previous analysis. The MPS emulator remains computationally more expensive in most cases, particularly for smaller system sizes or when the required bond dimension becomes large, resulting in occasional time peaks. This increased cost arises from the higher entanglement generated by the QAOA ansatz, which demands more resources to maintain accuracy in the MPS representation. However, as the number of nodes increases and the ansatz depth grows, the relative performance gap narrows, and in some instances, the MPS emulator even surpasses the statevector approach in efficiency. This suggests that for larger and more complex systems, the MPS method may begin to offer scalability advantages, as also observed in Subfigures 5.14c and 5.14d.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

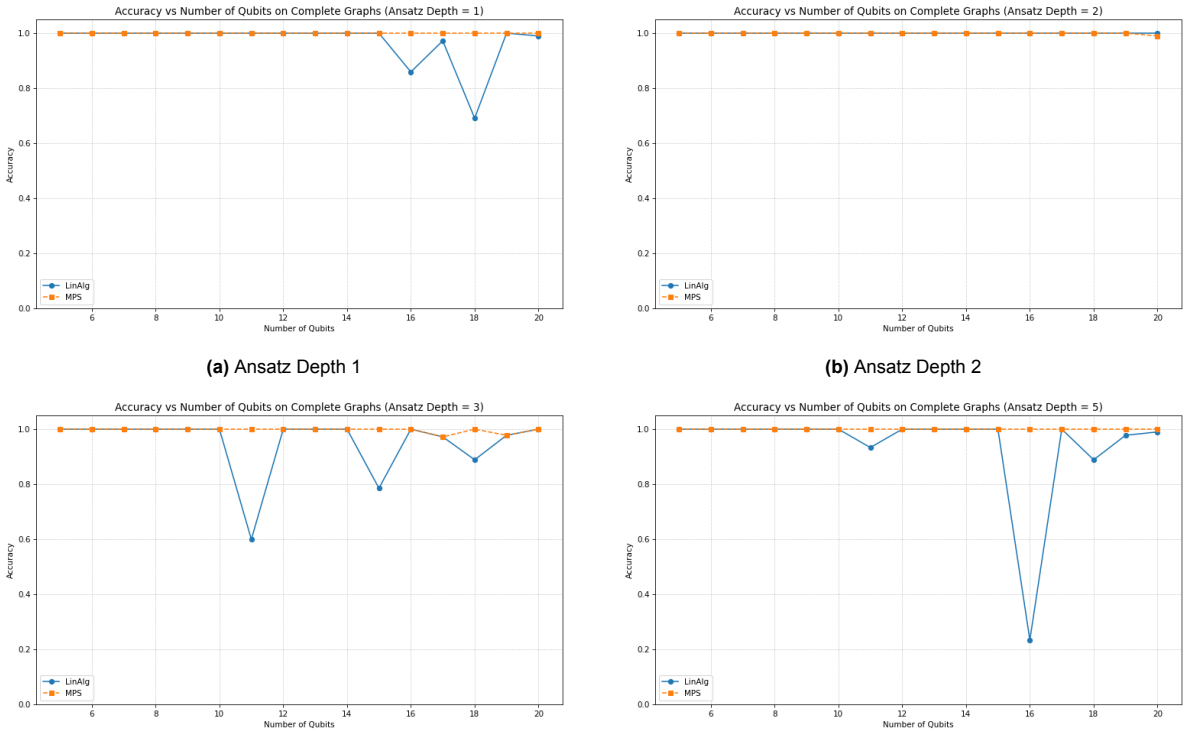
(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.14: Execution time comparison for different ansatz depths on 4-regular graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

5.3.3. Complete Graphs

We conclude this comparison with the final set of instances, corresponding to complete graphs. The results are striking: both emulators achieve near-perfect solutions across all ansatz depths and instances, highlighting once again that higher connectivity facilitates QAOA in reaching optimal outcomes. Importantly, this increased connectivity does not appear to hinder the MPS emulator. Moreover, the MPS approach continues to demonstrate greater robustness compared to the LinAlg emulator, which exhibits occasional drops in performance for specific instances, whereas the MPS results remain consistently reliable.



(a) Ansatz Depth 1

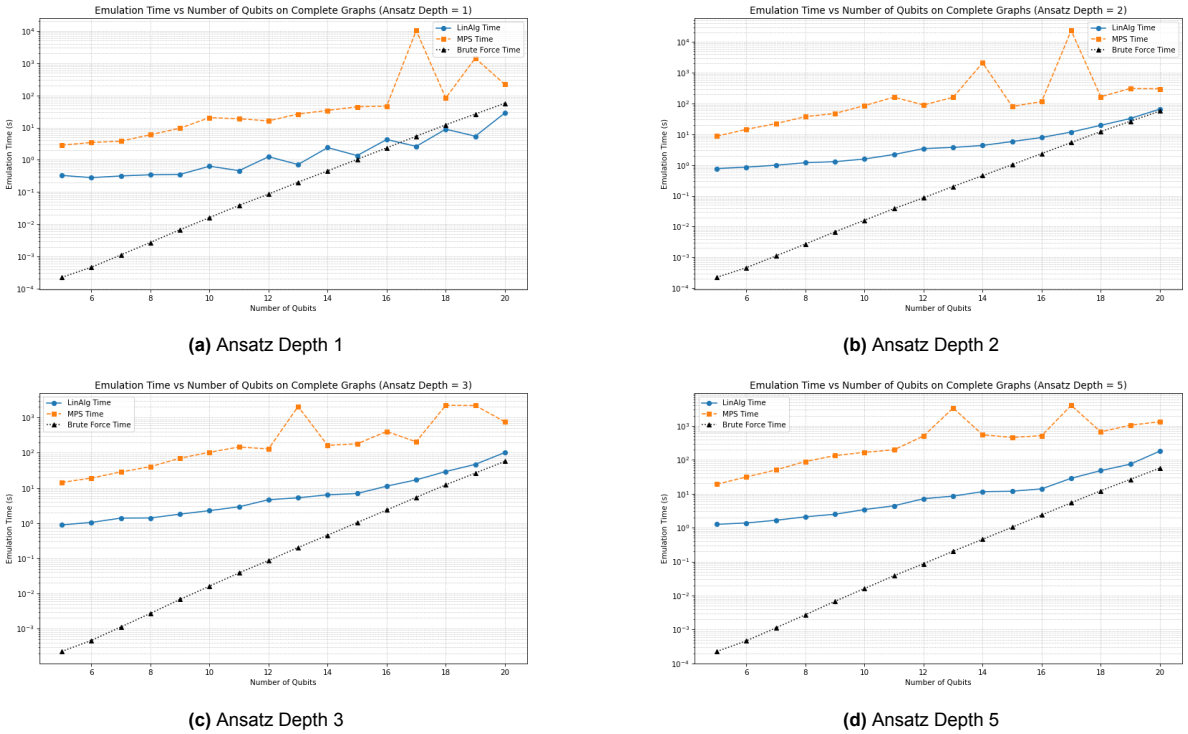
(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.15: Accuracy comparison for different ansatz depths on complete graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

We conclude this comparison by examining the execution time of both emulators. As expected, the MPS approach remains more computationally demanding than the LinAlg method. However, in this case, no clear time advantage is observed for either emulator across the considered instances and depths. Notably, the scaling of execution time appears to plateau for the highly connected complete graphs. This is particularly interesting, as one might have anticipated even longer emulation times given the increased times observed from the 2-regular to 4-regular graphs.



(a) Ansatz Depth 1

(b) Ansatz Depth 2

(c) Ansatz Depth 3

(d) Ansatz Depth 5

Figure 5.16: Execution time comparison for different ansatz depths on complete graphs using statevector and MPS emulators. Each subfigure corresponds to a specific depth.

5.4. Statevector against MPS emulator in Cheng Random Circuits

In this section, we evaluate the performance of the statevector and MPS emulators on random quantum circuits generated following the procedure described in Section 2. The benchmark set consists of circuits with sizes ranging from 5 to 60 qubits and depths between 5 and 75 layers, increasing in steps of 5. Each circuit is emulated using both emulators to identify the parameter regimes where one approach outperforms the other in terms of computational efficiency.

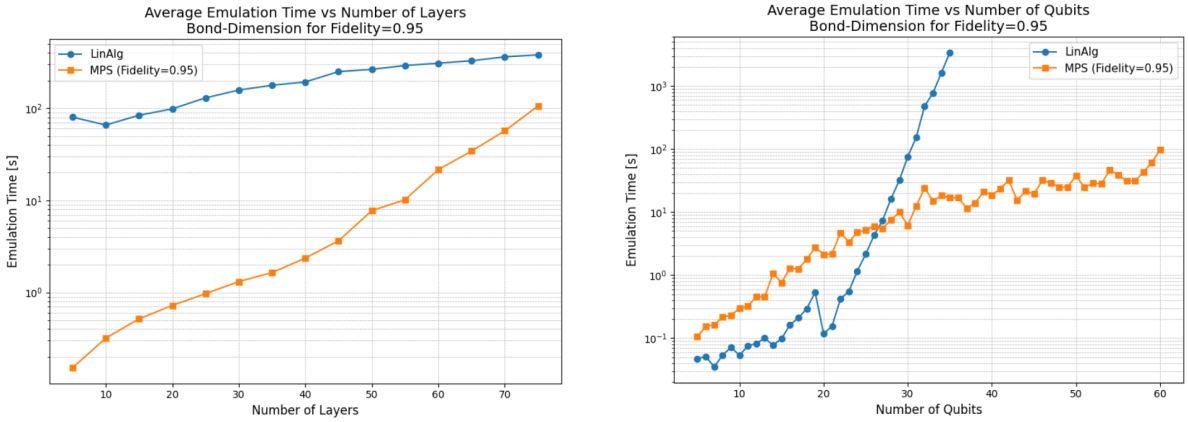
For the MPS emulator, we start each emulation with a bond dimension of 2 and increase it by increments of 2 until reaching a truncation fidelity of at least 0.95.

5.4.1. Computational Time Comparison

Figure 5.17 summarizes the computational performance of the LinAlg and MPS emulators across circuit depth and qubit number.

In Figure 5.17a, the average computational time, over the number of qubits, is plotted against the number of layers in the circuits. The figure shows that LinAlg is significantly more computationally expensive than MPS for shallow circuits with few layers. However, as the number of layers increases, the gap between the two emulators quickly decreases, since the computational cost of the MPS emulator grows more rapidly with circuit depth. For sufficiently deep circuits, LinAlg's full-precision emulation can become less time-consuming than MPS, making it the more efficient option in that regime.

Figure 5.17b presents the average computational time, over all depths, as a function of the number of qubits. For small circuits, MPS is initially slower, and LinAlg outperforms it. A crossover point is observed around 27 qubits, indicating that for circuits of general depths beyond this size, MPS becomes the faster emulator. Beyond this point, the LinAlg-based emulator exhibits exponential scaling, making emulation increasingly impractical and reaching practical limits at approximately 35 qubits within the considered experimental setup. In contrast, MPS continues to scale more gracefully, successfully reaching up to 60 qubits, albeit with a significant increase in runtime (from hundreds to thousands of seconds across the range). These results do not account for potential performance improvements from parallelization, which may mitigate runtime growth but were not explored in this work.



(a) Average computational time vs number of layers for *LinAlg* and MPS emulation.

(b) Average computational time vs number of qubits for LinAlg and MPS emulators.

Figure 5.17: Comparison of average computational time for LinAlg and MPS emulators across (a) number of layers and (b) number of qubits.

Figure 5.18 presents a heatmap summarizing the performance of LinAlg versus MPS emulators across all sampled circuits. Each square represents a single circuit, with colors on a logarithmic scale indicating the ratio of LinAlg runtime to MPS runtime: blue corresponds to faster LinAlg, grey to roughly equal runtimes, and red to faster MPS.

For circuits with a low number of qubits (up to 22), LinAlg consistently outperforms MPS by up to one order of magnitude. In the intermediate regime of 22–27 qubits—consistent with the trends observed in previous figures—both emulators exhibit comparable runtimes, with a tradeoff in which circuits with more qubits and shallower depths favor MPS. For larger circuits, MPS becomes increasingly advantageous, particularly for shallow circuits, achieving speedups of up to three orders of magnitude. However, for deeper circuits in this regime, LinAlg remains competitive, illustrating that depth and qubit number jointly determine the optimal emulator choice.

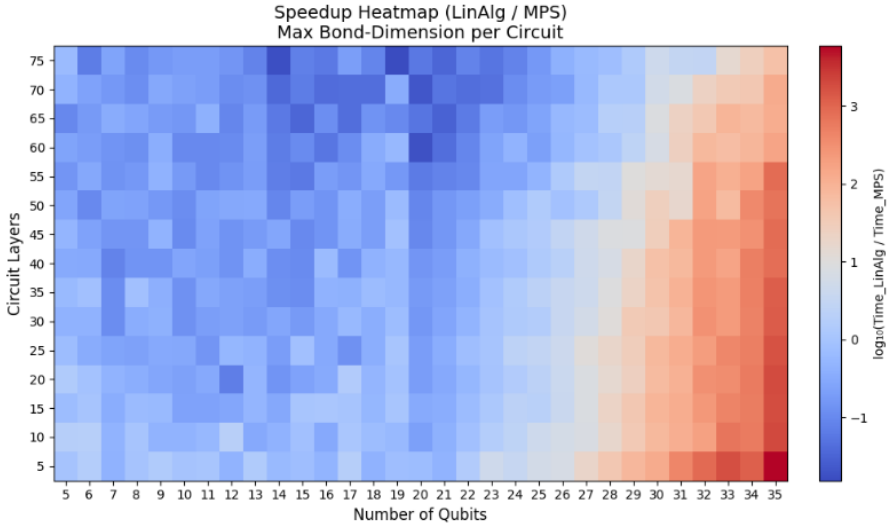


Figure 5.18: Speedup heatmap: logarithmic ratio of LinAlg to MPS computational time. Red/orange denotes regions of clear MPS advantage.

Figure 5.19 illustrates the evolution of the MPS bond dimension across all circuits used in our performance study. For shallow circuits (e.g., 5 layers), only very small bond dimensions are required to achieve high fidelity, largely independent of the number of qubits. This indicates that, in low-depth regimes, circuit size has minimal impact on the required bond dimension. However, as circuit depth in-

creases, the number of qubits begins to play a significant role. For instance, at a depth of 75 layers, the required bond dimension is below 10 for a 5-qubit circuit but grows to approximately 350 for a 60-qubit circuit.

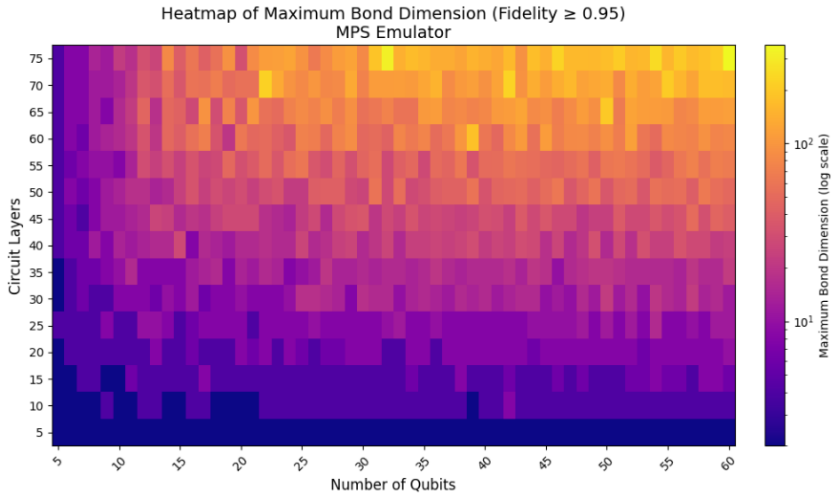


Figure 5.19: Heatmap showing the bond dimension required to reach 0.95 fidelity in MPS emulation across all circuits of varying depth and qubit number.

5.4.2. Entanglement Study

To better understand how fidelity in MPS emulations depends on entanglement, we analyzed the Von Neumann entanglement across the natural cuts of the circuit (i.e., cuts between qubit 1 and the rest, {1,2} and the rest, and so on), resulting in an array of entanglement values for each circuit (i.e, a distribution).

In Figures 5.20a and 5.20b, we show two representative plots for a circuit with 26 qubits and 60 layers. In both plots, one line (orange) illustrates the evolution of the MPS fidelity as the bond dimension increases. In blue, we show the corresponding L^2 distance[1].

In Figure 5.20a, the L^2 distance is computed between the MPS entanglement distribution at each bond dimension and the exact distribution obtained from the LinAlg emulator. In Figure 5.20b, the distance is instead computed between the entanglement distributions of consecutive bond dimensions.

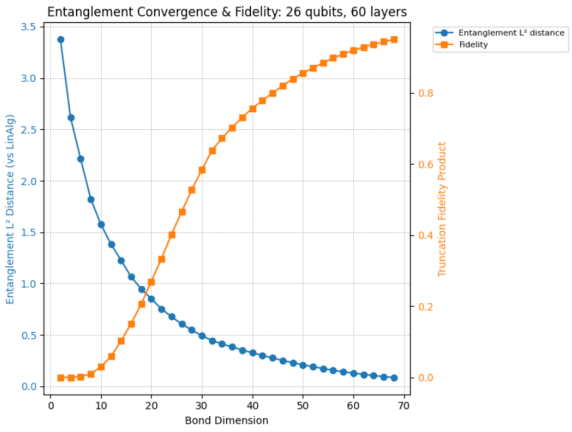
As expected, fidelity and distance are inversely correlated: larger distances correspond to lower fidelity, while as the distances converge toward zero—indicating that increasing the bond dimension no longer significantly changes the entanglement distribution—the fidelity growth slows down and approaches 1. This behavior highlights that the convergence of the entanglement distribution is a good indicator of the bond dimension needed to achieve high-fidelity MPS emulations. We observed that this trend holds for all cases, however, we were not able to find an analytical relation for this phenomena.

Finally, Figure 5.21 provides a direct visualization of how the entanglement distribution evolves as the MPS bond dimension increases for the 26-qubit, 60-layer circuit. Each colored line represents a different bond dimension, while the black line corresponds to the exact entanglement distribution obtained from the LinAlg emulator. As the bond dimension grows, the MPS distribution gradually approaches the exact profile, and the differences between successive bond dimensions become smaller. This illustrates that the gap in the entanglement profile slows down with increasing bond dimension, reflecting the diminishing returns in fidelity growth observed in Figures 5.20a and 5.20b.

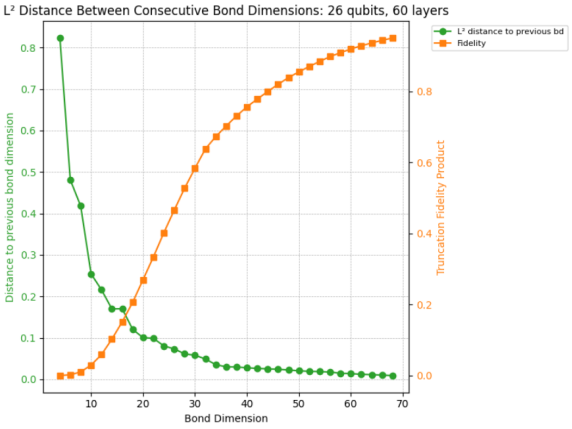
[1]The L^2 distance between two entanglement distributions $\vec{S}_1$ and $\vec{S}_2$ is defined as

$$L^2(\vec{S}_1, \vec{S}_2) = \sqrt{\sum_{i=1}^{N-1} \left(S_1^{(i)} - S_2^{(i)}\right)^2},$$

where N is the number of qubits and $S^{(i)}$ denotes the von Neumann entanglement across cut i.



(a) L^2 distance (blue) between and Fidelity (orange) between the MPS entanglement distribution and the exact distribution from LinAlg as a function of bond dimension.



(b) L^2 distance (blue) between and Fidelity (orange) consecutive bond dimension MPS runs as a function of bond dimension.

Figure 5.20: Analysis of MPS fidelity versus entanglement convergence for a 26-qubit, 60-layer circuit. (a) Shows the evolution of fidelity and the L^2 distance to the exact entanglement distribution from LinAlg. (b) Shows the evolution of fidelity and the L^2 distance between consecutive bond dimension runs.

Figure 5.21: Evolution of the von Neumann entanglement distributions across natural cuts for a 26-qubit, 60-layer circuit as the MPS bond dimension increases. Each colored line corresponds to a different bond dimension, while the black line indicates the exact entanglement distribution calculated with the LinAlg emulator.

6

Conclusions and Future Directions

In this work, we set out to explore how QAOA compares with its recursive variant, RQAOA, across different graph topologies, how noise affects their performance, and how statevector and MPS emulators scale with circuit size and depth. Our experiments in the Qaptiva Appliance showed that while RQAOA is known to outperform QAOA on complete graphs for the MAX-CUT problem, this advantage does not hold for general graphs such as 2-regular or 4-regular graphs. Nevertheless, RQAOA exhibits a more robust trend in accuracy across all depths and graph types. This robustness likely comes from its iterative approach, which allows the algorithm to improve poor intermediate solutions during the process. Even though RQAOA is usually more computationally expensive than standard QAOA, It exhibits more favorable scaling with respect to both the number of nodes and the ansatz depth. Despite this, both QAOA and RQAOA remain significantly slower and less accurate than classical methods for the graph instances we considered.

When incorporating noise in statevector-based emulations, we found that even for small instances, the computational cost increased dramatically, making the emulation of medium and large systems essentially infeasible. This overhead is mainly due to how noise is processed in the emulation environment, rather than gate times, as we did not see time spikes for deeper or bigger circuits. In terms of accuracy, the various types of noise did not lead to systematically worse solutions, but rather to mostly chaotic outcomes. In some cases, noise even helped outperform the noiseless case, possibly because it smoothed the cost landscape, giving the optimizer a better chance to avoid poor local minima. These effects were consistent across all graph topologies we tested.

Moving to the comparison of emulators, the results were somewhat surprising. The MPS emulator often outperformed the exact statevector emulator in terms of accuracy, which partially aligns with the findings of [86], where it was observed that for one-layer QAOA circuits, reducing the bond dimension mainly lowers the contrast of the cost landscape without significantly shifting the location of the minimum. This effect seems to hold even for deeper circuits in the types of graphs we studied. One factor that may contribute to this improved accuracy is that we selected the best results across different bond dimensions, which benefits MPS by giving it more chances to perform well. On the other hand, MPS emulations were often slower than statevector emulations, and in some cases, the difference in runtime was substantial. However, for larger systems and deeper circuits, MPS scaled better, suggesting that it could be the preferred emulator when the statevector approach becomes exponentially expensive.

We also explored pseudo-random circuits up to 60 qubits and 75 layers. Here, MPS was generally more efficient for shallow circuits, aligning with the literature, and started to lose efficiency for deeper circuits for this type of circuits. For small qubit numbers, statevector remained more efficient, up to about 27 qubits, but beyond roughly 35 qubits, it becomes impractical, whereas MPS could handle up to 60 qubits. Interestingly, for shallow circuits (for example, 5 layers), only very small bond dimensions were needed to achieve high fidelity, largely independent of the number of qubits, indicating that circuit size has minimal impact on required bond dimensions in low-depth regimes. We also attempted to relate Von Neumann entanglement entropy to MPS fidelity. While we did not find a clear, generalizable

relationship, we did observe that convergence of the entanglement distribution to the exact distribution correlates with higher fidelity, suggesting a potential method for estimating the bond dimension needed for accurate emulations.

One of the most remarkable, and initially unexpected, findings of this work is that QAOA performs better on graphs with higher connectivity for MAX-CUT problems. This counterintuitive trend was consistent across all emulation methods, including statevector, MPS, RQAOA, and noisy emulations, and aligns with observations from [87].

For future research, there are several directions that could build on these results. Studying different families of circuits could help understand which structures benefit most from MPS emulation. Extending the QAOA and RQAOA analysis to more diverse graph topologies, beyond just regular graphs, could provide further insights into algorithm robustness and performance trends. Continuing to explore the relationship between entanglement and MPS fidelity could offer practical guidelines for choosing bond dimensions and improving emulation efficiency.

Overall, this work studied how QAOA, RQAOA, and the emulators perform and scale, looked at the effect of noise, identified cases where MPS can be more efficient, and pointed out directions for future research.

References

[1] Daowen Qiu. “Chapter 1 - Development history and potential applications of quantum computing. This book has a companion website hosting complementary materials.” In: *Theoretical Foundations of Quantum Computing*. Ed. by Daowen Qiu. Morgan Kaufmann, 2026, pp. 1–10. DOI: https://doi.org/10.1016/B978-0-44-327704-7.00006-2.

[2] Muhammed Golec et al. “Chapter 1 - Quantum computing at a glance”. In: *Quantum Computing*. Ed. by Rajkumar Buyya and Sukhpal Singh Gill. Morgan Kaufmann, 2025, pp. 3–18. DOI: https://doi.org/10.1016/B978-0-443-29096-1.00010-6.

[3] Dilip Paneru et al. “Entanglement: quantum or classical?” In: *Reports on Progress in Physics* 83.6 (May 2020), p. 064001. DOI: 10.1088/1361-6633/ab85b9.

[4] Thomas Beck et al. “Integrating quantum computing resources into scientific HPC ecosystems”. In: *Future Generation Computer Systems* 161 (2024), pp. 11–25. DOI: https://doi.org/10.1016/j.future.2024.06.058.

[5] McKinsey & Company. *The Year of Quantum: From Concept to Reality in 2025*. Accessed: 2025-10-07. 2025.

[6] Shi-Yao Hou et al. *SpinQ Gemini: a desktop quantum computer for education and research*. 2021. arXiv: 2101.10017 [quant-ph].

[7] Ali Javadi-Abhari et al. *Quantum computing with Qiskit*. 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[8] Cirq Developers. *Cirq*. Zenodo, Aug. 2025. DOI: 10.5281/ZENODO.4062499.

[9] Quantinuum. *Quantinuum: Full stack Quantum Computing Company*. https://www.quantinuum.com/. 2021.

[10] Xanadu Quantum Technologies. *PennyLane — Quantum Machine Learning Library*. https://pennylane.ai/. 2024.

[11] Yilun Zhao et al. “7 - Efficient full-state simulation for quantum AI systems: Efficient full-state simulation for quantum AI systems”. In: *Quantum Computational AI*. Ed. by Long Cheng, Nishant Saurabh, and Ying Mao. Morgan Kaufmann, 2025, pp. 135–152. DOI: https://doi.org/10.1016/B978-0-44-330259-6.00016-5.

[12] Jacob Biamonte and Ville Bergholm. *Tensor Networks in a Nutshell*. 2017. arXiv: 1708.00006 [quant-ph].

[13] M. C. Bañuls et al. “Simulation of many-qubit quantum computation with matrix product states”. In: *Phys. Rev. A* 73 (2 Feb. 2006), p. 022344. DOI: 10.1103/PhysRevA.73.022344.

[14] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2000.

[15] The CUDA-Q development team. *CUDA-Q*. https://github.com/NVIDIA/cuda-quantum. Version 0.9.1. 2024. DOI: 10.5281/zenodo.14503457.

[16] Tyson Jones et al. “QuEST and High Performance Simulation of Quantum Computers”. In: *Scientific Reports* 9.1 (2019), p. 10736. DOI: 10.1038/s41598-019-47174-9.

[17] Eviden. *Eviden Qaptiva800: Quantum Emulation Appliance*. Accessed: 2025-09-23. 2025.

[18] J.A. Bravo-Montes, Miriam Bastante, and Cyril Allouche. “Quantum Emulation for High-Performance Computing Centers: Qaptiva Hpc”. In: *2025 Annual Modeling and Simulation Conference (ANNSIM)*. 2025, pp. 1–13.

[19] F. Pan et al. *Efficient quantum circuit simulation by tensor network methods on modern GPUs*. arXiv preprint. arXiv:2310.03978. 2023.

[20] P. Seitz et al. “Simulating quantum circuits using tree tensor networks”. In: *Quantum* 7 (2023), p. 964. DOI: 10.22331/q-2023-03-30-964.

[21] M. F. Mor-Ruiz and W. Dür. *Noisy stabilizer formalism*. arXiv preprint. arXiv:2212.08677. 2022.

[22] Y. Li and H. Miao. *Quantum multiple-valued decision diagrams with linear transformations*. arXiv preprint. arXiv:2207.11395. 2022.

[23] R. Wille, S. Hillmich, and L. Burgholzer. “Decision diagrams for quantum computing”. In: *Design Automation and Test in Europe Conference and Exhibition (DATE 2023)*. Springer, 2023. DOI: 10.1007/978-3-031-22842-4_5.

[24] Guifre Vidal. “Efficient classical simulation of slightly entangled quantum computations”. In: *Physical Review Letters* 91.14 (2003), p. 147902. DOI: 10.1103/PhysRevLett.91.147902.

[25] Ulrich Schollwöck. “The density-matrix renormalization group in the age of matrix product states”. In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192. DOI: 10.1016/j.aop.2010.09.012.

[26] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A Quantum Approximate Optimization Algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014). Submitted November 14, 2014.

[27] Eunok Bae and Soojoon Lee. “Recursive QAOA outperforms the original QAOA for the MAX-CUT problem on complete graphs”. In: *Quantum Information Processing* 23.3 (Feb. 2024). DOI: 10.1007/s11128-024-04286-0.

[28] Song Cheng et al. “Simulating noisy quantum circuits with matrix product density operators”. In: *Phys. Rev. Res.* 3 (2 Apr. 2021), p. 023005. DOI: 10.1103/PhysRevResearch.3.023005.

[29] Andres Bravo et al. *A Methodology to Select and Adjust Quantum Noise Models Through Emulators: Benchmarking Against Real Backends*. 2024. DOI: 10.21203/rs.3.rs-4575191/v1.

[30] Robert B. Griffiths. *Consistent Quantum Theory*. Cambridge University Press, 2002. Chap. 9. DOI: 10.1017/CBO9780511606351.010.

[31] Quantum Computing at Davis. *The Qubit*. https://qc-at-davis.github.io/QCC/How-Quantum-Computing-Works/The-Qubit/The-Qubit.html. Accessed: 2025-11-25.

[32] Gyeongju Song et al. “A Parallel Quantum Circuit Implementations of LSH Hash Function for Use with Grover’s Algorithm”. In: *Applied Sciences* 12.21 (2022). DOI: 10.3390/app122110891.

[33] Haoyu Liao and Qingbin Luo. *Quantum Circuit Synthesis for AES with Low DW-cost*. Cryptology ePrint Archive, Paper 2025/1494. 2025.

[34] Michał Stęchły. *Introduction to Variational Quantum Algorithms*. 2024. arXiv: 2402.15879 [quant-ph].

[35] J. C. Slater. “The Normal State of Helium”. In: *Phys. Rev.* 32 (3 Sept. 1928), pp. 349–360. DOI: 10.1103/PhysRev.32.349.

[36] Abhinav Kandala et al. “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets”. In: *Nature* 549.7671 (Sept. 2017), pp. 242–246. DOI: 10.1038/nature23879.

[37] Jeffrey C. Lagarias, Bjorn Poonen, and Margaret H. Wright. “Convergence of the Restricted Nelder–Mead Algorithm in Two Dimensions”. In: *SIAM Journal on Optimization* 22.2 (Jan. 2012), pp. 501–532. DOI: 10.1137/110830150.

[38] M. Powell. “A View of Algorithms for Optimization Without Derivatives”. In: *Mathematics TODAY* 43 (Jan. 2007).

[39] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[40] Han Qi et al. “Variational quantum algorithms: fundamental concepts, applications and challenges”. In: *Quantum Information Processing* 23 (2024). DOI: 10.1007/s11128-024-04438-2.

[41] Jarrod R. McClean et al. “Barren plateaus in quantum neural network training landscapes”. In: *Nature Communications* 9.1 (2018), p. 4812.

[42] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. “Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms”. In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070. DOI: 10.1002/qute.201900070.

[43] Abhinav Anand et al. “A quantum computing view on unitary coupled cluster theory”. In: *Chemical Society Reviews* 51.5 (2022), pp. 1659–1684. DOI: 10.1039/d1cs00932j.

[44] Aidan Pellow-Jarman et al. “The effect of classical optimizers and Ansatz depth on QAOA performance in noisy devices”. In: *arXiv preprint arXiv:2307.10149* (2023).

[45] Oumayma Bouchmal et al. “From classical to quantum machine learning: survey on routing optimization in 6G software defined networking”. In: *Frontiers in Communications and Networks* 4 (Nov. 2023). DOI: 10.3389/frcmn.2023.1220227.

[46] Joseph Emerson et al. “Pseudo-Random Unitary Operators for Quantum Information Processing”. In: *Science* 302.5653 (2003), pp. 2098–2100. DOI: 10.1126/science.1090790. eprint: https://www.science.org/doi/pdf/10.1126/science.1090790.

[47] S. Boixo et al. “Characterizing quantum supremacy in near-term devices”. In: *Nature Physics* 14.7 (2018), pp. 595–600. DOI: 10.1038/s41567-018-0124-x.

[48] Stephan Tornier. *Haar Measures*. 2020. arXiv: 2006.10956 [math.GR].

[49] Holger H. Hoos and Thomas Stützle. “1 - INTRODUCTION”. In: *Stochastic Local Search*. Ed. by Holger H. Hoos and Thomas Stützle. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2005, pp. 13–59. DOI: https://doi.org/10.1016/B978-155860872-6/50018-4.

[50] Hui-Min Li et al. “Ising Hamiltonians for Constrained Combinatorial Optimization Problems and the Metropolis-Hastings Warm-Starting Algorithm”. In: *Advanced Quantum Technologies* 6.9 (July 2023). DOI: 10.1002/qute.202300101.

[51] Benjamin Grimmer. *Max-Cut Polytopes and their Approximations*. Tech. rep. Available at https://www.ams.jhu.edu/~grimmer/MaxCut.pdf (accessed 2025-11-25). Johns Hopkins University, Department of Applied Mathematics & Statistics, 2022.

[52] Ryszard Horodecki et al. “Quantum entanglement”. In: *Reviews of Modern Physics* 81.2 (June 2009), pp. 865–942. DOI: 10.1103/revmodphys.81.865.

[53] Shikhar Uttam. “Chapter 4 - Introduction to Quantum Information Processing”. In: *Quantum Information Processing and Quantum Error Correction*. Ed. by Ivan Djordjevic. Oxford: Academic Press, 2012, pp. 119–144. DOI: https://doi.org/10.1016/B978-0-12-385491-9.00004-6.

[54] Claude E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[55] Martin B. Plenio and Shashank Virmani. “An Introduction to Entanglement Measures”. In: *quant-ph/0504163* (2005).

[56] S.-Y. Hou et al. “SpinQ Gemini: A desktop quantum computer for education and research”. In: *arXiv preprint arXiv:2101.10017* (2021).

[57] A. Zulehner and R. Wille. “Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 90–95.

[58] R. Fitzpatrick. *Normalization of the wavefunction*. LibreTexts. 2025.

[59] LUMI. *Quantum simulation resource requirements*. CSC – IT Center for Science. 2025.

[60] A. Berezutskii et al. “Tensor networks for quantum computing”. In: *arXiv preprint arXiv:2025* (2025).

[61] P. Seitz et al. “Simulating quantum circuits using tree tensor networks”. In: *Quantum* 7 (2023), p. 964.

[62] M. F. Mor-Ruiz and W. Dür. “Noisy stabilizer formalism”. In: *arXiv preprint arXiv:2212.08677* (2022).

[63] S. Aaronson and D. Gottesman. “Improved simulation of stabilizer circuits”. In: *Physical Review A* 70.5 (2004), p. 052328.

[64] Y. Li and H. Miao. “Quantum multiple-valued decision diagrams with linear transformations”. In: *arXiv preprint arXiv:2207.11395* (2022).

[65] R. Wille, S. Hillmich, and L. Burgholzer. *Decision diagrams for quantum computing*. Springer, 2023.

[66] G. E. Santoro et al. “Theory of quantum annealing of an Ising spin glass”. In: *Science* 295.5564 (2002), pp. 2427–2430.

[67] B. Heim et al. “Quantum versus classical annealing of Ising spin glasses”. In: *Science* 348.6231 (2015), pp. 215–217.

[68] In: *Quantum Information Theory*. Cambridge University Press, Nov. 2016, pp. xi–xii. DOI: 10.1017/9781316809976.001.

[69] Pau Escofet et al. “An accurate and efficient analytic model of fidelity under depolarizing noise oriented to large scale quantum system design”. In: *Quantum Science and Technology* 10.3 (July 2025), p. 035061. DOI: 10.1088/2058-9565/aded2f.

[70] Anna Leonteva et al. *Comparative Benchmarking of Utility-Scale Quantum Emulators*. 2025. arXiv: 2504.14027 [quant-ph].

[71] Timothy Proctor et al. “Measuring the capabilities of quantum computers”. In: *Nature Physics* 18.1 (Dec. 2021), pp. 75–79. DOI: 10.1038/s41567-021-01409-7.

[72] J. Eisert, M. Cramer, and M. B. Plenio. “Colloquium: Area laws for the entanglement entropy”. In: *Reviews of Modern Physics* 82.1 (Feb. 2010), pp. 277–306. DOI: 10.1103/revmodphys.82.277.

[73] Y. Hirata et al. “An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture”. In: *Quantum Information and Computation VII*. Vol. 7340. Proc. SPIE. 2009, 73400Y. DOI: 10.1117/12.820387.

[74] Gushu Li, Yufei Ding, and Yuan Xie. “Tackling the qubit mapping problem for NISQ-era quantum devices”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’19)*. 2019, pp. 1001–1014. DOI: 10.1145/3297858.3304023.

[75] Alwin Zulehner, Alexandru Paler, and Robert Wille. “Efficient mapping of quantum circuits to the IBM QX architectures”. In: *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE ’18)*. 2018, pp. 1135–1138. DOI: 10.23919/DATE.2018.8342148.

[76] Alireza Shafaei, Robert Wille, and Rolf Drechsler. “Quantum circuits optimizing synthesis for linear nearest neighbor architectures”. In: *Proceedings of the 2013 Design Automation Conference (DAC ’13)*. 2013, pp. 1–6. DOI: 10.1145/2463209.2488812.

[77] Cisco Systems, Inc. *Cisco AnyConnect Secure Mobility Client*. Version 4.10 or latest available at the time of writing. 2023.

[78] Mobatek. *MobaXterm*. 2023.

[79] Michael J. McGuffin, Jean-Marc Robert, and Kazuki Ikeda. “How to Write a Simulator for Quantum Circuits from Scratch: A Tutorial”. In: *arXiv preprint arXiv:2506.08142v1* (2025). Section 2.

[80] *Qaptiva 800s Appliance Documentation*. Accessed: 2025-09-30. Atos.

[81] Steven R. White. “Density matrix formulation for quantum renormalization groups”. In: *Physical Review Letters* 69.19 (1992), pp. 2863–2866. DOI: 10.1103/PhysRevLett.69.2863.

[82] Shi-Ju Ran et al. *Tensor Network Contractions: Methods and Applications to Quantum Many-Body Systems*. Jan. 2020. DOI: 10.1007/978-3-030-34489-4.

[83] Korbinian Kottmann. *Introducing matrix product states for quantum practitioners*. Last updated September 22, 2025. 2024. URL: https://pennylane.ai/qml/demos/tutorial_mps.

[84] Masoud Mohseni et al. *How to Build a Quantum Supercomputer: Scaling from Hundreds to Millions of Qubits*. 2025. arXiv: 2411.10406 [quant-ph].

[85] Maxime Oliva. *An entanglement-aware quantum computer simulation algorithm*. 2023. arXiv: 2307.16870 [quant-ph].

[86] Maxime Dupont et al. “Calibrating the Classical Hardness of the Quantum Approximate Optimization Algorithm”. In: *PRX Quantum* 3.4 (Dec. 2022). DOI: 10.1103/prxquantum.3.040339.

[87] Elisabeth Wybo and Martin Leib. “Missing Puzzle Pieces in the Performance Landscape of the Quantum Approximate Optimization Algorithm”. In: *Quantum* 9 (Oct. 2025), p. 1892. DOI: 10.22331/q-2025-10-22-1892.