UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

telecos
BCN

Quantum Simulation

Bachelor's Degree Thesis
Submitted to the Faculty at the
Escola Tècnica Superior
d'Enginyeria de Telecomunicació de Barcelona
of the Universitat Politècnica de Catalunya
by

Albert López Escudero

In partial fulfillment
of the requirements for the
Degree in Electronics Engineering

Advisor: Vincenzo de Maio & Ivona Bandric
Reviewer: Javier Rodrigez Fonollosa

Barcelona, January 2025

ETSETB – Degree in Electronics Engineering

Resum

Cada exemplar del Treball de Fi de Grau (TFG) ha de contenir un Resum, que és un breu extracte del TFG. En termes d'estil, el Resum hauria de ser una versió reduïda del projecte: una introducció concisa, un compendi dels resultats i les principals conclusions o arguments presentats en el projecte. El Resum no ha de superar les 150 paraules i cal que estigui traduït al català, castellà i anglès.

Resumen

Cada ejemplar del Trabajo de Fin de Grado (TFG) debe incluir un Resumenque es un breve extracto del TFG. En cuanto al estilo, el Resumen debería ser una versión reducida del proyecto: una introducción breve, un resumen de los resultados principales y las conclusiones o argumentos principales presentados en el proyecto. El Resumen no debe exceder las 150 palabras y debe estar traducido al catalán, castellano e inglés.

Summary

Each copy of the Bachelor's Thesis (TFG) must include a Summary, which is a concise abstract of the TFG. In terms of style, the Summary should be a condensed version of the project: a brief introduction, a summary of the main results, and the conclusions or key arguments presented in the project. The Summary should not exceed 150 words and must be translated to catalan, spanish and english.

A Dedication page may be included in your thesis just before the Acknowledgments page, but it is not a requirement.

Acknowledgements

It is appropriate, but not mandatory, to declare the extent to which assistance has been given by members of the staff, fellow students, technicians or others in the collection of materials and data, the design and construction of apparatus, the performance of experiments, the analysis of data, and the preparation of the thesis (including editorial help). In addition, it is appropriate to recognize the supervision and advice given by your advisor.

Revision history and approval record

Revision	Date	Author(s)	Description
1.0	dd/mm/yyyy	AME	Document creation
1.1	dd/mm/yyyy	AME, JPV	Error correction
2.0	dd/mm/yyyy	AME, MLO	Revised after review
4.0	dd/mm/yyyy	AME	Final version

DOCUMENT DISTRIBUTION LIST

Role	Surname(s) and Name
[Student]	
[Project Supervisor 1]	
[Project Supervisor 2 (if applicable)]	

Written by:		**Reviewed and approved by:**	
Date	dd/mm/yyyy	Date	dd/mm/yyyy
Name	Xxxxxxx Yyyyyyy	Name	Xxxxxxx Yyyyyyy
Position	Project Author	Position	Project Supervisor

Contents

Summary . . . 2
Acknowledgements . . . 4
Revision history and approval record . . . 5
Contents . . . 6
List of figures . . . 9
List of tables . . . 10
Abbreviations . . . 11

1 Introduction **12**
1.1 Work goals . . . 13
1.2 Requirements and specifications . . . 13
1.3 Methods and procedures . . . 14
1.4 Work plan . . . 15

2 State of the Art **16**
2.1 Quantum Simulation . . . 16
2.2 Key Concepts in Quantum Mechanics . . . 17
2.2.1 Qubit . . . 18
2.2.2 Quantum Superposition . . . 19
2.2.3 Quantum Decoherence . . . 19
2.3 The Hamiltonian in Quantum Mechanics . . . 20
2.3.1 Mathematical Definition . . . 20
2.3.2 Role in the Schrödinger Equation . . . 21
2.3.3 Hamiltonian in Multi-Particle Systems . . . 21
2.3.4 Importance in Quantum Simulations . . . 21
2.4 VQE: Variational Quantum Eigensolver . . . 22
2.4.1 Fundamental Principles and Stages of the Algorithm . . . 22
2.4.2 Advantages and Challenges . . . 23
2.4.3 Outlook in Quantum Simulation . . . 23
2.5 Different Ansätze in Quantum Chemistry and Quantum Computing . . . 23
2.5.1 Hartree–Fock-based Ansätze (Classical Reference) . . . 24
2.5.2 Unitary Coupled Cluster (UCC) . . . 25
2.5.3 Problem-Inspired or Custom Ansätze . . . 25
2.6 Optimizers . . . 25

2.6.1 Gradient Descent (GD) . . . 26
2.6.2 Momentum Optimizer . . . 26
2.6.3 Nesterov Momentum Optimizer (NMomentum) . . . 26
2.6.4 RMSProp . . . 27
2.6.5 Adagrad . . . 27
2.6.6 Adam . . . 27
2.6.7 Quantum Natural Gradient (QNG) . . . 28
2.6.8 Importance of Optimizers in Quantum Simulation . . . 28

3 Methodology / project development 29
3.1 Tools and Frameworks Selection . . . 29
3.2 Project Structuring . . . 30
3.2.1 Code Organization . . . 30
3.2.2 Main Directory . . . 31
3.2.3 `config/` Directory . . . 31
3.2.4 `modules/` Directory . . . 31
3.2.5 `temp_results_autograd/` Directory . . . 31
3.3 Development and Implementation . . . 32
3.3.1 Hamiltonian Construction Process . . . 33
3.3.2 Adaptive Ansatz Construction and Operator Selection . . . 34
3.3.3 Cost Function Definition . . . 35
3.3.4 Mixed Electronic and Geometric Optimization Strategy . . . 36
3.4 Limitations and Mitigation Measures . . . 39
3.5 Version Control . . . 39

4 Results 41
4.1 Experiments and Tests . . . 41
4.2 Experimentos y Pruebas . . . 41
4.3 Comparación de Interfaces . . . 41
4.3.1 Optimización y Registro de Tiempos . . . 42
4.3.2 Tiempo de Cómputo por Función . . . 43
4.3.3 Conclusiones . . . 44
4.4 Comparación del Ansatz . . . 44
4.5 Optimizador . . . 46
4.5.1 Selección del Optimizador . . . 46
4.5.2 Selección del Step Size . . . 48
4.5.3 Numero de Iteraciones . . . 49

5 Sustainability Analysis and Ethical Implications 50
5.1 Sustainability Matrix . . . 50
5.1.1 Environmental Perspective . . . 50
5.1.2 Economic Perspective . . . 52
5.1.3 Social Perspective . . . 53
5.2 Ethical Implications . . . 54
5.3 Relation to the Sustainable Development Goals (SDGs) . . . 54

6 Conclusions and Future Work **55**
6.1 Conclusions 55
6.2 Future Directions 55

A Logic Gates **58**
A.1 Simple Logic Gates 58
A.2 Multi-Qubit Logic Gates 59

List of Figures

1.1 Gantt diagram showing the project timeline and the distribution of tasks over the development period. 15

2.1 Comparison of computation time for molecular simulations: classical vs quantum. 18

4.1 Tiempo de ejecución de las distintas moléculas según la iteración en la simulación. 42
4.2 Tiempo de ejecución en distintas partes del código. 43
4.3 Energy vs Time for different ansatz and iterations. 45
4.4 Energy vs Time for different ansatz. 45
4.5 Selección óptima de *step size* para cada optimizador y molécula. 46

List of Tables

4.1 Tiempos de ejecución para distintas moléculas e interfaces, utilizando *Gradient Descent*. 42
4.2 Comparación de tiempos de ejecución entre las interfaces JAX y autograd para diferentes moléculas. 43
4.3 Energía final y tiempo de optimización para H_2O (sin usar FCI). 47
4.4 Energía final y tiempo de optimización para H_2. 47
4.5 Energía final y tiempo de optimización para LiH. 48

Abbreviations

ETSETB Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

EU European Union

GEF Grau en Enginyeria Física

GREELEC Grau en Enginyeria Electrònica de Telecomunicació

GRETST Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació

CHAPTER 1

Introduction

In recent years, computing has undergone significant evolution, reaching a point where improving the hardware of traditional devices presents considerable challenges. This has driven research in quantum computing, a technology that promises to revolutionize the field by enabling much more efficient calculations and superior processing capabilities. In applications such as molecular simulation, quantum computing has proven to be remarkably more efficient than classical computing, justifying investment in its development.

However, one of the main obstacles of quantum computing is the high cost associated with its devices. To run programs on a quantum computer, it must operate at extremely low temperatures, close to 0.1 Kelvin, which significantly increases the cost and technical complexity. Therefore, methods are being investigated to improve and optimize these devices, making them more accessible and viable for broader use.

Currently, one of the most practical ways to harness quantum computing's potential is through **quantum simulation**. Quantum simulation allows us to study complex quantum systems using either quantum simulators or classical computers. By emulating the behavior of quantum systems, we can explore quantum algorithms and applications effectively without necessarily requiring a fully functional quantum computer.

The main objective of this project is to develop a quantum simulator capable of efficiently modeling molecules using advanced quantum algorithms, with a specific focus on implementing the Variational Quantum Eigensolver (VQE) algorithm. This algorithm combines quantum circuits for state preparation and classical optimizers that adjust parameters to minimize the system's energy. Additionally, the project introduces innovations in the design of adaptive quantum circuits, the integration of simultaneous optimizations of nuclear and electronic coordinates, all within a modular and extensible project structure that allows for easy and effective adaptation of the simulation.

1.1 Work goals

The primary goal of this project is to contribute a novel methodology to the field of molecular simulations by:

- Optimize the quantum simulation framework to minimize computational overhead while maintaining accuracy in modeling molecular systems, focusing on enhancing the Variational Quantum Eigensolver (VQE) efficiency.
- Develop an adaptive quantum ansatz capable of dynamically selecting and integrating operators with the highest impact on performance, ensuring rapid convergence and reduced computational cost.
- Integrate advanced hybrid optimization cycles that streamline both electronic state and nuclear geometry refinements, with an emphasis on computational speed and scalability.
- Design a high-performance, modular architecture tailored for extensibility, facilitating the efficient inclusion of cutting-edge algorithms, ansatz designs, and optimization strategies.

1.2 Requirements and specifications

To achieve these objectives, the following requirements and specifications have been established:

- **Dynamic Ansatz Construction**: Integrate an adaptive ansatz mechanism that employs energy gradient calculations to iteratively select and include the most impactful operators, reducing the complexity of the quantum circuits without compromising accuracy.
- **Advanced Hybrid Optimization**: Develop a unified optimization cycle combining variational parameter updates and nuclear geometry refinements, supported by precise gradient-based techniques. Ensure the use of optimizers for rapid convergence and enhanced stability.
- **Scalable Hamiltonian Management**: Implement a modular Hamiltonian construction process that accommodates different molecular geometries and basis sets with efficient computation of molecular properties.
- **Efficient Gradient Computation**: Optimize gradient calculations for both quantum and nuclear coordinates using automatic differentiation, enabling simultaneous updates and faster convergence.
- **Performance Monitoring and Visualization**: Develop tools for tracking energy convergence, execution time, and molecular geometry evolution. Include detailed visualizations (e.g., energy evolution plots, 3D geometries) to ensure transparency and allow performance analysis.

- **Extensibility and Modularity**: Ensure a modular project structure that facilitates the inclusion of new ansatz types, optimizers, and molecular systems with minimal adjustments. Employ a structured directory for clear separation of functionality and scalability.
- **Parallel Execution Capability**: Leverage multiprocessing capabilities to evaluate multiple optimizers and configurations in parallel, enabling comprehensive performance analysis across various configurations.

1.3 Methods and procedures

In our project, we have implemented a series of advanced methodologies that integrate quantum and classical techniques with specific innovations to address molecular optimization problems and energy calculations efficiently. We have combined quantum computing, which enables the evaluation of fundamental system properties such as energy and quantum states, with classical algorithms that optimize the parameters required to represent these states effectively. This hybrid approach, inspired by the ideas of Richard Feynman, has been validated in works like that of Arute et al. (2019), who achieved quantum supremacy using digital circuits to solve problems inaccessible to classical computers. Within this framework, we used the Variational Quantum Eigensolver (VQE) algorithm as the core of our simulation. This algorithm combines parameterized circuits, known as ansatz, with classical optimization to minimize the expected value of the molecular Hamiltonian, which describes the total energy of the system. Introduced by Peruzzo et al. in 2014, VQE has proven particularly effective for noisy intermediate-scale quantum (NISQ) devices.

Additionally, we have implemented an adaptive ansatz that enhances efficiency by dynamically constructing the quantum circuit. This ansatz iteratively selects relevant operators based on energy gradients, an approach inspired by the ADAPT-VQE algorithm by Grimsley et al. (2019), ensuring faster and more accurate convergence to the ground state. Since scalable quantum hardware is not available, we used classical simulators such as PennyLane's default.qubit to emulate digital quantum circuits. This has allowed us to design and evaluate algorithms robustly in classical environments while maintaining the ability to test innovative approaches.

Among the main innovations implemented, we developed a dynamic adaptive ansatz that incorporates only the quantum operators with the greatest impact on energy, calculated through gradients using automatic differentiation. This not only optimizes computational efficiency but also reduces the complexity of the model. We also implemented a hybrid optimization cycle that simultaneously adjusts the ansatz parameters and nuclear positions, accurately capturing the interaction between electrons and molecular geometry. The modular structure of the code has been key to enabling the incorporation of new functionalities and the comparison of different optimization strategies without affecting other components of the system. Finally, we designed the system to support multiple types of ansatz and optimizers, facilitating an exhaustive comparative analysis

and ensuring the scalability of the project for future extensions in quantum chemistry and molecular simulations.

1.4 Work plan

During the development of the project, the initial plan was largely followed. However, several complications arose that extended some deadlines, requiring adjustments to the overall timeline to achieve the established objectives.

The main setback was related to the integration of the JAX interface. From the outset, we were confident that using JAX would significantly improve the performance of the simulations. However, upon completing the initial integration, we were surprised to find that the performance not only failed to improve but actually worsened. This unexpected result led us to conduct more checks than initially planned, aiming to verify the accuracy of the results and identify the root cause of this behavior.

This situation caused delays in the second phase of the project, which, in turn, required a reduction in the time allocated to the third phase to meet the deadlines. Despite this, we decided not to compromise on the quality of the project. As a result, the overall development time had to be extended, increasing the daily hours dedicated to the project. This additional effort ensured that the initial objectives were achieved without compromising the proposed quality standards.

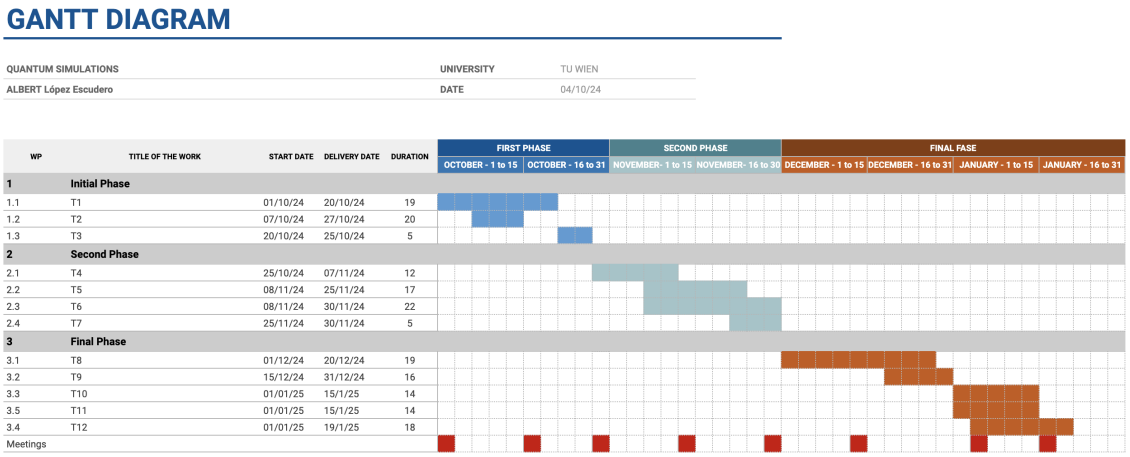


Figure 1.1: *Gantt diagram showing the project timeline and the distribution of tasks over the development period.*

CHAPTER 2

State of the Art of the Technology Used or Applied in this Thesis

In this project, we focus on the operation of quantum simulators applied to molecular simulation, an area where quantum computing offers significant advantages. The main objective is to develop a program capable of simulating different molecules in a simple and efficient manner.

To this end, we will review the basic concepts of quantum mechanics that are essential to understand the fundamentals and potential of quantum computing, and explore the techniques of quantum simulation that allow us to harness quantum computing capabilities even with current technological limitations.

2.1 Quantum Simulation

Quantum simulation has emerged as an advanced and essential technique for studying complex quantum systems, especially those that are inaccessible or present great challenges for direct analysis using classical methods. Based on the proposal of Richard Feynman, who postulated that a computer built from quantum elements could overcome the limitations of classical computers in simulating quantum phenomena, quantum simulation has progressed significantly. It encompasses both digital and analog simulations and has expanded its applicability in various scientific areas.

There are mainly two approaches in quantum simulation: **Digital Quantum Simulation (DQS)** and **Analog Quantum Simulation (AQS)**. DQS employs the quantum circuit model, where systems are represented by qubits that evolve through quantum gates to reproduce the dynamics of the target system. This approach is universal, as it can, in principle, simulate any quantum system, although not always efficiently. On the other hand, AQS involves creating a quantum system that directly emulates the Hamiltonian of the system under study, allowing certain properties of the simulated system, such as time evolution, to be reproduced approximately. This method is particularly useful when a qualitative representation is required rather than high precision.

In addition to these approaches, there are algorithms inspired by quantum information theory that facilitate the classical simulation of quantum systems. Techniques such as **Matrix Product States (MPS)** and **Projected Entangled Pair States (PEPS)** allow representing particle systems on classical computers more efficiently than standard classical methods, optimizing the calculation of properties of complex quantum systems.

The applications of quantum simulation are broad and encompass multiple scientific fields. In condensed matter physics, it allows the study of models such as the Hubbard model and quantum phase transitions, fundamental for understanding phenomena like superconductivity. In quantum chemistry, it facilitates the calculation of molecular energies and complex chemical reactions. In high-energy physics and cosmology, it emulates particles in high-energy fields and cosmological phenomena. Furthermore, quantum simulation is instrumental in the analysis of open quantum systems and in the investigation of quantum chaos, allowing exploration of interactions with the environment and chaotic dynamics in the quantum realm.

However, quantum simulation faces significant challenges related to the precise control of the quantum simulator systems and the management of decoherence and errors, which can affect the accuracy of the results. The amount of required resources, such as the number of qubits and quantum gates, also depends on the size and complexity of the system to be simulated. It is estimated that quantum simulators require between 40 and 100 qubits to surpass the computational power of classical computers in specific problems. Despite these challenges, technological advances continue to improve the viability and efficiency of quantum simulation, promising to transform research in natural sciences and expand our understanding of quantum phenomena.

2.2 Key Concepts in Quantum Mechanics

It is essential to understand the difference between bits in classical computing and qubits in quantum computing to delve into this new technological paradigm.

In classical computing, the basic unit of information is the **bit**, which can take the value of 0 or 1. These bits are the foundation upon which conventional computers operate, processing information through combinations of these binary states.

In contrast, quantum computing uses the **qubit** or quantum bit as its basic unit. Unlike the classical bit, a qubit can exist in a superposition of states, meaning it can simultaneously represent the values 0 and 1 thanks to the principle of superposition in quantum mechanics. This property, along with phenomena such as quantum entanglement and interference, allows quantum computers to process information exponentially more efficiently for certain problems.

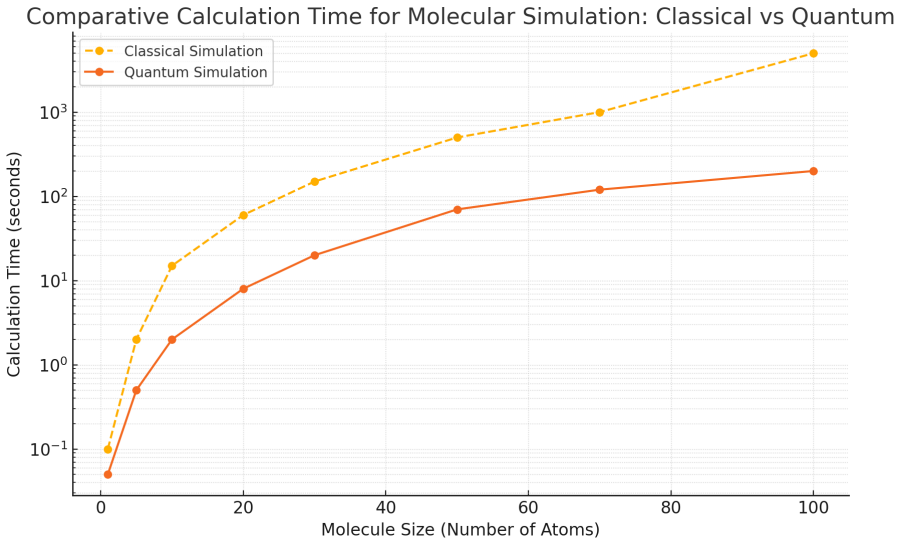


Figure 2.1: *Comparison of computation time for molecular simulations: classical vs quantum.*

Understanding how qubits operate and their differences from classical bits is essential to appreciate the revolutionary potential of quantum computing.

2.2.1 Qubit

The **qubit** is the basic unit of information in quantum computing. While the classical bit can only be in one of two states (0 or 1), a qubit can be in a superposition of both states simultaneously. This is due to the principle of quantum superposition, one of the fundamental characteristics of quantum mechanics.

Mathematically, a qubit is represented as a linear combination of the basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers that satisfy the normalization condition $|\alpha|^2+|\beta|^2 = 1$. These coefficients indicate the probability amplitudes of finding the qubit in the states $|0\rangle$ or $|1\rangle$ upon measurement.

In addition to superposition, qubits can exhibit **quantum entanglement**, a property that allows creating strong correlations between qubits that cannot be explained by classical physics. Entanglement is essential for the computational power of quantum computers, as it enables processing and storing an exponentially larger amount of information than classical systems.

For example, while a classical system of n bits can represent one of 2^n possible state combinations, a quantum system of n qubits can represent a superposition of all those

combinations simultaneously. This capability is what allows quantum computers to tackle complex problems more efficiently.

However, manipulating and maintaining qubits is a significant technical challenge. Qubits are extremely sensitive and can be affected by interactions with the environment, leading to **quantum decoherence**. To minimize this effect and preserve quantum properties, it is necessary to keep systems in controlled conditions, such as very low temperatures, close to absolute zero.

2.2.2 Quantum Superposition

Quantum superposition allows a quantum system to exist in multiple states simultaneously until a measurement is performed. This characteristic is key to the functioning of quantum computers, as it enables processing a large amount of information in parallel.

In quantum systems, superposition is combined with **quantum interference**, where the probability amplitudes of states can reinforce or cancel each other out. This phenomenon is exploited in quantum algorithms to increase the probability of obtaining the correct result. For example, in Grover's algorithm, constructive interference amplifies the probability of the desired state, significantly improving the efficiency of searching for elements in an unsorted database.

Superposition is especially useful in simulating complex molecular systems. Quantum computers can naturally model the superpositions of electronic states in molecules, which is crucial for studying chemical reactions and molecular properties that are difficult to address with classical methods due to the exponential growth of computational resources required.

2.2.3 Quantum Decoherence

Quantum decoherence is one of the main challenges in quantum computing. It refers to the loss of a system's quantum properties, such as superposition and entanglement, due to unwanted interactions with the environment. This loss causes the quantum system to transition toward classical behavior, affecting the accuracy and reliability of quantum calculations.

Qubits are extremely sensitive to external disturbances, such as electromagnetic fluctuations, vibrations, and temperature changes. These interactions can cause quantum states to mix with those of the environment, leading to a loss of coherence that is irreversible and degrades the stored quantum information.

To mitigate the effects of decoherence, various strategies are implemented:

- **System Isolation**: Designing physical systems that minimize unwanted interactions with the environment, using materials and techniques that protect qubits from external disturbances.

- **Quantum Error Correction**: Implementing error correction codes that allow detecting and correcting errors without directly measuring the qubit's state, thereby preserving quantum information.
- **Dynamic Control**: Applying techniques such as pulse refocusing and dynamic pulse sequences that actively compensate for disturbances and extend the coherence time of qubits.

Controlling and mitigating decoherence are essential for the advancement of quantum computing and its application in areas like molecular simulation, where the precision of calculations is fundamental.

2.3 The Hamiltonian in Quantum Mechanics

The Hamiltonian is a fundamental concept originating from classical mechanics, introduced by William Rowan Hamilton in 1833. Hamiltonian mechanics is a reformulation of classical mechanics that provides powerful tools for studying the dynamics of systems. The Hamiltonian function represents the total energy of the system, expressed in terms of generalized coordinates and momenta, and is given by the sum of the kinetic and potential energies.

In quantum mechanics, the **Hamiltonian operator** plays a central role in describing the energy and time evolution of quantum systems. It represents the total energy of the system, including both kinetic and potential energies, and is essential for formulating the Schrödinger equation.

2.3.1 Mathematical Definition

The Hamiltonian operator, commonly denoted as $\hat{H}$, is a self-adjoint operator acting on the Hilbert space associated with the quantum system. For a single particle in one dimension, the Hamiltonian is expressed as:

$$\hat{H} = \hat{T} + \hat{V}$$

where:

- $\hat{T}$ is the kinetic energy operator.
- $\hat{V}$ is the potential energy operator.

In terms of the position $\hat{x}$ and momentum $\hat{p}$ operators, these are defined as:

$$\hat{T} = \frac{\hat{p}^2}{2m} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2}$$

$$\hat{V} = V(\hat{x})$$

Here, m is the mass of the particle, $\hbar$ is the reduced Planck constant, and $V(\hat{x})$ is the potential energy function depending on position.

2.3.2 Role in the Schrödinger Equation

The Hamiltonian is central to the Schrödinger equation, which describes how the quantum state of a system evolves over time. The time-dependent Schrödinger equation is expressed as:

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle$$

where $|\psi(t)\rangle$ is the state vector of the system at time t. For time-independent systems, the Schrödinger equation reduces to the eigenvalue equation:

$$\hat{H}|\psi\rangle = E|\psi\rangle$$

Here, E represents the eigenvalues of the Hamiltonian, corresponding to the allowed energy levels of the system, and $|\psi\rangle$ are the associated eigenstates.

2.3.3 Hamiltonian in Multi-Particle Systems

For systems with multiple particles, the Hamiltonian includes additional terms representing interactions between particles. For example, for a system of two particles, the Hamiltonian is expressed as:

$$\hat{H} = \hat{T}_1 + \hat{T}_2 + \hat{V}_1 + \hat{V}_2 + \hat{V}_{12}$$

where:

- $\hat{T}_1$ and $\hat{T}_2$ are the kinetic energy operators of particles 1 and 2, respectively.
- $\hat{V}_1$ and $\hat{V}_2$ are the individual potential energy operators.
- $\hat{V}_{12}$ represents the potential interaction between the two particles.

2.3.4 Importance in Quantum Simulations

In quantum simulations, especially in algorithms like the Variational Quantum Eigensolver (VQE), the Hamiltonian is decomposed into a sum of simpler terms, often expressed in terms of Pauli operators. This decomposition facilitates implementation on quantum circuits and allows estimating the system's energy through measurements on qubits.

Understanding the structure and properties of the Hamiltonian is essential for modeling and simulating quantum systems, as it determines the possible energies and dynamics of the system under study.

2.4 VQE: Variational Quantum Eigensolver

Among the hybrid quantum-classical algorithms developed to address quantum simulation challenges, the *Variational Quantum Eigensolver (VQE)* has gained particular relevance. This method seeks to approximate the ground-state energy of a target Hamiltonian, such as the electronic Hamiltonian of a molecule, by efficiently combining quantum state preparation with classical optimization techniques.

2.4.1 Fundamental Principles and Stages of the Algorithm

VQE is founded on the **variational principle** of quantum mechanics, which states that the expectation value of the energy $E(\vec{\theta})$ for any normalized trial state $|\psi(\vec{\theta})\rangle$ is always an upper bound to the true ground-state energy E_0:

$$E(\vec{\theta}) = \langle \psi(\vec{\theta}) | \hat{H} | \psi(\vec{\theta}) \rangle \quad \geq \quad E_0.$$

Because $E(\vec{\theta})$ depends on a set of parameters $\vec{\theta}$, the VQE algorithm iteratively updates these parameters to minimize $E(\vec{\theta})$. Once no further reduction in $E(\vec{\theta})$ is possible, the algorithm identifies the lowest value reached as an approximation of E_0.

In practice, VQE proceeds in a loop that integrates quantum and classical resources:

1. **Quantum State Preparation:** A parameterized quantum circuit, commonly termed an *ansatz*, is constructed with a set of adjustable parameters $\vec{\theta}$. This circuit leverages unitary gates (e.g., single-qubit rotations, entangling gates) to create a trial wavefunction:
$$|\psi(\vec{\theta})\rangle .$$

2. **Measurement of the Expected Energy:** The Hamiltonian $\hat{H}$ is decomposed into a sum of Pauli operators acting on the qubits. The expectation value $\langle \psi(\vec{\theta}) \mid \hat{H} \mid \psi(\vec{\theta}) \rangle$ is obtained by measuring the appropriate Pauli operators on the quantum device, typically requiring multiple circuit executions due to non-commuting terms.

3. **Classical Optimization:** The measured energy serves as a cost function, $E(\vec{\theta})$. A classical optimizer—such as *Gradient Descent* or *Adam*—then updates the parameters $\vec{\theta}$ to minimize this cost.

4. **Iteration and Convergence:** Steps 1–3 repeat until a convergence criterion is satisfied, for instance,
$$\left| E\left(\vec{\theta}_{k+1}\right) - E\left(\vec{\theta}_k\right) \right| \; < \; \delta,$$
where δ is a small threshold for energy differences. The final set $\vec{\theta}^*$ yields an approximate ground-state wavefunction and energy.

2.4.2 Advantages and Challenges

VQE has garnered extensive interest in fields like **quantum chemistry**, **materials science**, and even combinatorial optimization problems mapped to Hamiltonians. Its hybrid structure allows leveraging near-term quantum devices (with limited qubits and gate depths) while offloading resource-intensive tasks—such as parameter updates—to classical computers.

Despite its promise, VQE faces several practical hurdles:

- **Noise and Decoherence:** Real quantum devices suffer from errors that deteriorate the fidelity of prepared states and measurements, requiring error-mitigation strategies and noise-aware ansatz designs.
- **Barren Plateaus:** High-dimensional parameter spaces can contain large regions where gradients vanish, complicating the search for global minima.
- **Measurement Overhead:** Decomposing a Hamiltonian into many Pauli terms demands running multiple circuits, increasing sampling time and exposure to hardware noise.
- **Circuit Depth:** Accurate ansätze for complex systems may require deep circuits that quickly exceed the coherence times of current quantum processors.

2.4.3 Outlook in Quantum Simulation

Continuous improvements in both hardware (qubit quality, gate fidelity, and error-correction schemes) and software (advanced ansätze, better optimizers, error mitigation) keep driving VQE toward practical applications. Recent strategies such as *ADAPT-VQE*, which adaptively builds up an excitation operator set, and domain-specific ansätze integrated with error mitigation methods, further enhance VQE's accuracy for molecular systems.

As the number of qubits grows and quantum hardware matures, VQE is likely to become a central approach for tackling classically intractable problems in quantum chemistry, materials science, and beyond. Its flexible hybrid nature will continue to serve as a testbed for new optimization algorithms, ansatz designs, and measurement strategies, bridging current *Noisy Intermediate-Scale Quantum (NISQ)* devices with the longer-term ambition of fault-tolerant quantum computing.

2.5 Different Ansätze in Quantum Chemistry and Quantum Computing

In the context of variational quantum algorithms and quantum chemistry, an **ansatz** is a carefully chosen, often physically motivated, parametric form of the quantum state used to approximate the ground (or excited) state of a system described by a given Hamiltonian. The term *ansatz* originates from the German word "approach" or "initial guess," and it

reflects the central idea that we propose a functional form (or circuit structure) for the wavefunction and then optimize the parameters in search of the lowest possible energy.

Within the framework of the Variational Quantum Eigensolver (VQE), the ansatz is implemented as a parameterized quantum circuit whose gates depend on a set of continuous variables $\vec{\theta}$. By measuring the expectation value of the Hamiltonian with respect to this trial state, one obtains an energy estimate $E(\vec{\theta})$ which is then iteratively minimized by a classical optimizer. The success of a VQE calculation hinges critically on the expressiveness and resource requirements (number of gates, circuit depth, etc.) of the chosen ansatz.

Several ansätze have been proposed to achieve a balance between accuracy and computational cost. Below, we summarize the most relevant approaches, highlighting their theoretical underpinnings and current usage in quantum simulation.

2.5.1 Hartree–Fock-based Ansätze (Classical Reference)

A historically important "classical" ansatz in quantum chemistry arises from the **Hartree–Fock (HF)** approximation. In this method, the total wavefunction is assumed to be a single Slater determinant constructed from one-particle orbitals. Although it captures the fundamental antisymmetry required by the Pauli exclusion principle (i.e., fermionic exchange), it neglects most of the electron correlation. Post-HF methods, such as Configuration Interaction (CI), Many-Body Perturbation Theory (MPn), and Coupled Cluster (CC), then build on this reference state by introducing additional terms that account for electron correlation.

- **Configuration Interaction (CI):** Expands the wavefunction in a basis of Slater determinants (excitations) beyond the HF reference. The Full CI approach is exact within the chosen basis but scales exponentially with system size. Truncated CI methods (CIS, CID, CISD, etc.) reduce the computational cost but still grow quickly with system size.
- **Coupled Cluster (CC):** Expresses the wavefunction via an exponential of excitation operators acting on the HF reference. Formally written as

  $$|\Psi_{\mathrm{CC}}\rangle = e^{\hat{T}}|\Phi_{\mathrm{HF}}\rangle,$$

  where $\hat{T}$ is the sum of cluster excitation operators (singles, doubles, triples, etc.). Although Coupled Cluster with Singles and Doubles (CCSD) is often accurate, further inclusion of triples and higher excitations can be required for strongly correlated systems.

In the context of classical methods, these **ansätze** serve as trial wavefunctions whose coefficients are optimized using high-performance classical algorithms. Their conceptual basis—constructing physically motivated trial states that capture crucial features of the system—carries over into quantum computing.

2.5.2 Unitary Coupled Cluster (UCC)

A key adaptation of the Coupled Cluster theory to quantum computing is the **Unitary Coupled Cluster (UCC)** ansatz. It modifies the standard CC exponential by making it explicitly unitary:

$$|\Psi_{\mathrm{UCC}}\rangle = e^{\hat{T}(\vec{\theta})-\hat{T}^{\dagger}(\vec{\theta})}\,|\Phi_{\mathrm{HF}}\rangle,$$

where $\hat{T}(\vec{\theta})$ is typically truncated to include only single and double excitations (UCCSD). This approach guarantees that the resulting operator is unitary, which is crucial for hardware implementations in quantum computing since all gates must be unitary transformations.

- **UCCSD (Singles and Doubles):** The most widespread version of UCC is truncated at single and double excitations:

$$\hat{T}(\vec{\theta}) = \sum_{i,a} \theta_i^a \hat{a}_a^{\dagger}\hat{a}_i \;+\; \sum_{i,j,a,b} \theta_{i,j}^{a,b}\, \hat{a}_a^{\dagger}\hat{a}_b^{\dagger}\hat{a}_j\hat{a}_i \;+\; \cdots$$

  Here, i, j denote occupied orbitals and a, b virtual (unoccupied) orbitals. By exponentiating both $\hat{T}$ and $\hat{T}^{\dagger}$, the wavefunction stays normalized. However, the circuit depth can become large since implementing the exponential of a sum of non-commuting operators requires a trotterization or related approximation.

- **ADAPT-VQE and Variants:** To mitigate the high circuit cost, variants like *ADAPT-VQE* build up a UCC-type ansatz incrementally, selecting only those excitation operators that most significantly lower the energy at each step. This adaptive approach reduces the number of gates needed and often converges faster.

2.5.3 Problem-Inspired or Custom Ansätze

In some cases, **custom ansätze** are tailored to the specific physical or chemical system under investigation. For example, if certain symmetries (like particle number or spin) are known to be crucial for describing the ground state, one can design an ansatz that explicitly respects those symmetries. Such approaches can drastically reduce the parameter space and improve convergence, albeit with some additional effort in circuit design.

2.6 Optimizers

In quantum simulation algorithms such as the Variational Quantum Eigensolver (VQE), optimizers constitute a key element of the hybrid quantum-classical workflow. Their primary objective is to minimize the cost function

$$E(\vec{\theta}) = \langle \psi(\vec{\theta}) \mid \hat{H} \mid \psi(\vec{\theta}) \rangle,$$

where $\hat{H}$ represents the Hamiltonian of the system under study and $|\psi(\vec{\theta})\rangle$ is a parameterized quantum state, often referred to as the *ansatz*. After each quantum measurement, the optimizer updates the parameter vector $\vec{\theta}$ to guide the system toward the ground-state energy. This process is iterated until convergence, balancing the capabilities of quantum hardware with classical numerical techniques.

In the following subsections, we discuss the theoretical underpinnings and key features of several optimizers commonly employed in variational algorithms. While all these optimizers share the goal of efficiently navigating the parameter space, they differ in how they incorporate gradients, memory of past iterations, and adjustments of the learning rate.

2.6.1 Gradient Descent (GD)

Gradient Descent is one of the most fundamental methods for continuous optimization. At each iteration, it updates parameters by moving them in the direction opposite to the gradient of the cost function:

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \eta \nabla_{\vec{\theta}} E(\vec{\theta}_k),$$

where η is the learning rate and $\nabla_{\vec{\theta}} E(\vec{\theta})$ denotes the gradient of the cost function with respect to the parameters. Despite its simplicity, Gradient Descent can converge slowly or get trapped in local minima when dealing with complex or high-dimensional landscapes, making it less efficient if used alone in large-scale molecular simulations.

2.6.2 Momentum Optimizer

The *Momentum* method extends standard Gradient Descent by incorporating a velocity term that accumulates a fraction of previous updates. This approach can mitigate oscillations and speed up convergence in regions with shallow gradients. The update rule is given by:

$$\vec{v}_{k+1} = \gamma \vec{v}_k - \eta \nabla_{\vec{\theta}} E(\vec{\theta}_k),$$
$$\vec{\theta}_{k+1} = \vec{\theta}_k + \vec{v}_{k+1},$$

where γ (typically between 0.9 and 0.99) is the momentum coefficient that controls how much past gradients influence the current update. This optimizer often accelerates learning in practice by effectively smoothing noisy or rapidly changing gradients.

2.6.3 Nesterov Momentum Optimizer (NMomentum)

Nesterov Momentum, sometimes referred to as *Nesterov's Accelerated Gradient (NAG)*, refines the idea of Momentum by anticipating the next position of the parameters before computing the gradient. Concretely, one computes the gradient at $\vec{\theta} + \gamma \vec{v}$ rather than at

$\vec{\theta}$ only. The updates become:

$$\vec{v}_{k+1} = \gamma \vec{v}_k - \eta \nabla_{\vec{\theta}} E\big(\vec{\theta}_k + \gamma \vec{v}_k\big),$$
$$\vec{\theta}_{k+1} = \vec{\theta}_k + \vec{v}_{k+1}.$$

By 'looking ahead' in the direction of the velocity term, Nesterov Momentum tends to achieve smoother convergence and better performance on problems with numerous local minima or saddle points, which are common in complex quantum simulations.

2.6.4 RMSProp

RMSProp is a gradient-based optimizer that adaptively tunes the learning rate for each parameter by normalizing the gradient through a moving average of its recent magnitudes. This helps address issues of vanishing or exploding gradients, which can be particularly troublesome in variational circuits of moderate or large depth. Its core update equations are:

$$E[\nabla^2_{\vec{\theta}}]_k = \beta\, E[\nabla^2_{\vec{\theta}}]_{k-1} + (1-\beta)\, \nabla_{\vec{\theta}} E(\vec{\theta}_k)^2,$$
$$\vec{\theta}_{k+1} = \vec{\theta}_k - \eta\, \frac{\nabla_{\vec{\theta}} E(\vec{\theta}_k)}{\sqrt{E[\nabla^2_{\vec{\theta}}]_k + \epsilon}},$$

where $0 < \beta < 1$ is a decay factor controlling the smoothing effect, and ϵ is a small constant ensuring numerical stability.

2.6.5 Adagrad

Adagrad is an early approach to adaptive learning rates, designed to handle sparse or highly non-uniform gradients. It individually scales the updates by the inverse square root of the cumulative sum of gradients:

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \frac{\eta}{\sqrt{\sum_{i=1}^{k} \nabla_{\vec{\theta}} E(\vec{\theta}_i)^2 + \epsilon}}\, \nabla_{\vec{\theta}} E(\vec{\theta}_k).$$

This mechanism allows parameters with small but consistent gradients to receive larger updates, which can be helpful in certain quantum chemistry models where specific Hamiltonian terms dominate.

2.6.6 Adam

Adam (Adaptive Moment Estimation) has emerged as one of the most widely used optimizers in machine learning and, increasingly, in quantum algorithms. It combines Momentum-like accumulations of the first moment of gradients (i.e., the mean) with an RMSProp-like treatment of the second moment (i.e., the uncentered variance). The

update rules are:

$$\begin{aligned}
\vec{m}_{k+1} &= \beta_1 \vec{m}_k + (1-\beta_1)\nabla_{\vec{\theta}} E(\vec{\theta}_k), \\
\vec{v}_{k+1} &= \beta_2 \vec{v}_k + (1-\beta_2)\nabla_{\vec{\theta}} E(\vec{\theta}_k)^2, \\
\hat{\vec{m}}_{k+1} &= \frac{\vec{m}_{k+1}}{1-\beta_1^{k+1}}, \quad \hat{\vec{v}}_{k+1} = \frac{\vec{v}_{k+1}}{1-\beta_2^{k+1}}, \\
\vec{\theta}_{k+1} &= \vec{\theta}_k - \eta \frac{\hat{\vec{m}}_{k+1}}{\sqrt{\hat{\vec{v}}_{k+1}} + \epsilon},
\end{aligned}$$

where $0 < \beta_1, \beta_2 < 1$ are decay hyperparameters controlling how quickly the estimates of the first and second moments adjust. Adam's blend of adaptive step sizes and momentum often yields robust performance, even when the cost landscape is noisy or irregular, as is typical in quantum simulations.

2.6.7 Quantum Natural Gradient (QNG)

Unlike classical optimizers that rely on Euclidean metrics in parameter space, *Quantum Natural Gradient (QNG)* specifically incorporates the *Fubini–Study* metric, capturing how small changes in the parameters affect the underlying quantum state. By working with a geometry adapted to the quantum manifold, QNG can achieve faster and more reliable convergence in variational circuits. Conceptually, the update rule can be written as:

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \eta\, \boldsymbol{\mathcal{F}}^{-1} \nabla_{\vec{\theta}} E(\vec{\theta}_k),$$

where $\boldsymbol{\mathcal{F}}$ represents the *quantum Fisher information matrix*, a matrix encoding the local geometry of the parameterized state. Computing $\boldsymbol{\mathcal{F}}$ can be more demanding than classical gradients, but for many quantum chemistry or condensed-matter applications, the improved efficiency justifies this added cost.

2.6.8 Importance of Optimizers in Quantum Simulation

Optimizers bridge the gap between quantum hardware and classical processing by iteratively refining the variational parameters to minimize the expectation value of the Hamiltonian. They must contend with challenges specific to quantum simulation, such as measurement noise, limited qubit counts, and complex cost landscapes characterized by local minima and barren plateaus. Properly choosing and tuning the optimizer is paramount to achieving accurate, resource-efficient simulations. By harnessing the distinctive advantages of adaptive and momentum-based methods—as well as more specialized quantum-aware techniques like QNG—one can significantly improve the speed and reliability of variational algorithms, thereby pushing the capabilities of quantum simulation closer to practical applications in molecular modeling.

CHAPTER 3

Methodology / project development

In this chapter, the methodology used in the completion of the work will be detailed. Its aim is to offer a thorough account of the approaches and techniques used, ensuring replicability and academic rigor. It will not only cover the research methods and measurement techniques employed but will also delve into the specifics of software and hardware development. Whether the project involves qualitative analysis, quantitative measurements, computational modeling, or physical prototyping, this chapter should elucidate how each component contributes to the overall objectives.

In addition to describing the methods themselves, the chapter will also provide justifications for why specific methods were chosen over others. For example, it may explain the choice of a particular programming language, statistical test, or experimental setup. The chapter will also address the limitations of the methodology and how these have been mitigated or accounted for. Readers should come away with a clear understanding of how the project's development has been carried out, why certain choices were made, and how these methods serve to fulfill the initially established objectives.

3.1 Tools and Frameworks Selection

To achieve the objectives of our project, the first crucial step was to select the framework to be used throughout the development process. This decision was essential, as it directly influenced the progress and success of the project.

To make the decision on which framework to use, we compared the documentation of the two quantum simulation frameworks available in the market: PennyLane and Qiskit. These are the most comprehensive frameworks with similar features available at the time of creating this project. After reviewing the documentation, we ultimately chose to use PennyLane for two reasons.

The first reason was the amount of documentation related to quantum simulation. Once we started looking into how others were using these resources, we realized that in the field of molecular simulation, the existing documentation—both theoretical and especially

practical—was substantially greater. This provided us with more examples to begin developing our project and a more extensive theoretical background to understand the concepts we were working with.

The second reason for our choice was the frequent major changes implemented by Qiskit. We realized that while Qiskit is a tool that promises to be very good, it has historically undergone significant structural changes. For these reasons, this project has been developed using the PennyLane framework.

3.2 Project Structuring

After selecting the interface and implementing the initial version of the code, we reorganized the project to achieve a more modular structure. This revised organization not only makes it easier to add new functionalities but also ensures that the project can handle higher levels of complexity.

3.2.1 Code Organization

```
                    quantum_simulation_project/
config/
      config_functions.py: Configuration functions for the project.
      molecules.json: JSON file containing molecular data.
main.py: The main entry point for the program.
modules/
      ansatz_preparer.py: Quantum ansatz preparation.
      hamiltonian_builder.py: Molecular Hamiltonian construction.
      molecule_manager.py: Molecular data management.
      opt_mol.py: End-to-end molecular optimization.
      optimizer.py: Optimization algorithms.
      visualizer.py: Visualization tools.
temp_results_autograd/
      energy_evolution.png: Graph of energy convergence.
      filtered_report_autograd.txt: Filtered results report.
      final_geometries_3D.png: Final 3D geometries.
      nuclear_coordinates.png: Nuclear coordinates visualization.
      output.txt: Program output log.
      profile_output_autograd.txt: Autograd profiling output.
test/: Directory for tests.
```

Summary of Core Files:

3.2.2 Main Directory

- `main.py`: Central entry point of the program. It initializes the process by selecting molecules, configuring the optimizer, and setting up the ansatz. It also manages the optimization workflow, handles result storage, and produces comprehensive reports. Profiling tools evaluate computational performance.

3.2.3 `config/` Directory

- `config_functions.py`: Handles project configuration. This includes loading molecular data, selecting optimization algorithms, and setting initial parameters such as ansatz type and convergence tolerance. The module also allows adding custom molecules and organizing saved results.
- `molecules.json`: A structured JSON file containing information about molecules, including atomic symbols, coordinates, charges, and spin multiplicities.

3.2.4 `modules/` Directory

Core computational logic resides here:

- `ansatz_preparer.py`: Implements quantum circuit construction (ansätze) for both adaptive and traditional methods. Includes the UCCSD ansatz (single and double excitations) and hardware-efficient ansatzes featuring multiple circuit layers.
- `hamiltonian_builder.py`: Constructs the molecular Hamiltonian, a fundamental component of quantum simulations. Calculates Hartree-Fock reference states and can extract exact energy values from the Hamiltonian matrix.
- `molecule_manager.py`: Initializes molecules by processing atomic symbols, initial coordinates, and configuration parameters such as charge and multiplicity. Also computes important properties like the number of electrons and orbitals.
- `optimizer.py`: Contains optimization algorithms (e.g., Adam, QNG, RMSProp). Integrates parameter updates and nuclear coordinate adjustments in a unified optimization framework.
- `opt_mol.py`: Orchestrates the complete molecular optimization pipeline. Brings together Hamiltonian construction, molecule management, optimization routines, and result visualization.
- `visualizer.py`: Offers visualization tools for energy convergence and molecular geometries. Generates detailed graphical outputs in both linear and logarithmic scales.

3.2.5 `temp_results_autograd/` Directory

Contains intermediate results generated during simulations:

- `energy_evolution.png`: Graph of energy convergence over iterations.
- `nuclear_coordinates.png`: Visualization of nuclear coordinates during optimization.
- `filtered_report_autograd.txt`: Filtered report of relevant profiling metrics.
- `output.txt`: Primary output log of the program.

3.3 Development and Implementation

The *Variational Quantum Eigensolver* (VQE) was chosen as the primary method to estimate the ground state energy of the studied quantum system. VQE combines limited quantum processing (measurements and applicability in moderately deep circuits) with classical optimization techniques. Its selection is justified by:

- **Suitability for NISQ devices:** VQE is particularly well-suited for noisy intermediate-scale quantum (NISQ) devices, as it requires circuits of relatively low depth.
- **Flexible Ansatz:** It allows the use of various adaptive variational ansätze that capture essential electronic correlations.
- **Direct coupling to classical optimizers:** The VQE cost function (the expected energy) can be minimized with a wide range of classical methods, making it easy to experiment with different optimizers.

The core principle of VQE is the variational theorem, which guarantees that the expected energy of the ansatz is always an upper bound to the true ground state energy. By optimizing the ansatz parameters, the algorithm progressively approaches the actual energy minimum. We have already explained the concept of VQE in the state of the art chapter; now we will explain how we have implemented it in our project and how we have integrated it.

Principle of VQE: The VQE is based on the variational principle, which states that the expected energy of any approximate state $|\psi(\theta)\rangle$ is always greater than or equal to the real ground state energy E_0:

$$E(\theta) = \langle \psi(\theta)|H|\psi(\theta)\rangle \geq E_0$$

We have already discussed this concept, but it is necessary to emphasize it as it is the foundation of the entire algorithm. The idea is to find the parameters θ that minimize the expected energy, thereby approaching the real value of the ground state energy.

Next, we will detail how the VQE is implemented in our project, explaining how each component has been developed for trying to achieve the best performance and results.

3.3.1 Hamiltonian Construction Process

1. **Definition of Molecular Geometry:**

   Each molecule is defined by specifying its atomic symbols and Cartesian coordinates:

   Geometry Definition

   ```
   symbols = ['H', 'H']
   x_init = np.array([0.0, 0.0, 0.0,  # Hydrogen 1
                      0.0, 0.0, 0.74])  # Hydrogen 2
   ```

2. **Hamiltonian Construction:**

   The `build_hamiltonian` function creates the molecular Hamiltonian using PennyLane's quantum chemistry features. This function transforms the electronic Hamiltonian from second quantization into a qubit-based representation.

   Hamiltonian Build

   ```
   def build_hamiltonian(x, symbols, charge=0, mult=1, basis_name='sto
       -3g'):
       x = np.array(x)
       coordinates = x.reshape(-1, 3)
       hamiltonian, qubits = qml.qchem.molecular_hamiltonian(
           symbols, coordinates, charge=charge, mult=mult, basis=
               basis_name
       )
       h_coeffs, h_ops = hamiltonian.terms()
       h_coeffs = np.array(h_coeffs)
       return qml.Hamiltonian(h_coeffs, h_ops)
   ```

   Note: A basis set, such as `sto-3g`, is chosen to represent atomic orbitals. This predefined set of basis functions simplifies the simulation while retaining sufficient accuracy for many molecular systems.

   Function Description:

   - Transforms the electronic Hamiltonian into a Pauli-based qubit Hamiltonian suitable for quantum algorithms.
   - Incorporates user-defined net charge and spin multiplicity, enabling simulation of various electronic states.
   - Simplifies the computational space with a chosen basis set and, optionally, an active space to focus only on relevant orbitals and electrons.

3.3.2 Adaptive Ansatz Construction and Operator Selection

As we said before, the adaptive ansatz construction builds upon the conventional variational approach by strategically selecting only those excitations that offer the most significant energy reductions. Instead of starting from a large, fixed set of parameters, the algorithm begins with the Hartree-Fock state and incrementally introduces new excitations based on their calculated impact on lowering the system's energy. This methodology provides both theoretical and practical advantages in handling the complexity of the solution space.

The selection process begins with a predefined *operator pool*, typically composed of single and double excitation operators relevant to the molecular system. At the start of the procedure, no variational parameters are assigned, and the system is initialized in the reference Hartree-Fock state. At each iteration, the algorithm evaluates the energy gradients associated with adding each operator from the pool:

1. **Gradient Calculation:** For every candidate operator $\hat{O}_i$ in the pool, the partial derivative of the energy with respect to the parameter controlling $\hat{O}_i$ is computed. This step identifies how sensitive the energy is to introducing that particular excitation.

2. **Operator Ranking and Filtering:** All candidate excitations are ranked according to the absolute value of their gradients. Operators that produce negligible energy changes are discarded, while those offering substantial decreases are selected for inclusion.

3. **Incremental Ansatz Growth:** The selected operator(s) is then added to the ansatz. A new parameter is introduced and optimized, increasing the dimensionality of the parameter space *only where it matters*. This targeted expansion ensures that each additional parameter contributes meaningfully to lowering the energy.

4. **Pool Update and Iteration:** After adding the chosen operators, the process repeats. The operator pool is re-examined at subsequent steps, but it now excludes previously chosen operators unless they are included as parameterized parts of the ansatz. Over multiple iterations, the ansatz evolves adaptively, honing in on the most relevant subset of excitations.

A simplified code snippet, consistent with the project's structure, may appear as follows:

Adaptive Operator Selection

```
1 gradients = compute_operator_gradients(operator_pool,
      selected_excitations, params, hamiltonian, hf_state, dev,
      spin_orbitals)
2 selected_gate, max_grad_value = select_operator(gradients, operator_pool,
       convergence_threshold)
3 if selected_gate:
```

```
    selected_excitations.append(selected_gate)
    params = np.append(params, 0.0)  # Add new parameter for the chosen
        operator
    print(f"Added operator {selected_gate} with gradient {max_grad_value
        :.5e}")
else:
    print("No significant operators found. Convergence or local minimum
        reached.")
```

In this code, the `compute_operator_gradients` function evaluates each operator's gradient, while the `select_operator` function applies a filtering criterion based on a defined `convergence_threshold`. Only the most promising excitation is incorporated into the ansatz at each step, ensuring a controlled and meaningful expansion of the parameter space.

In numerical experiments, this targeted approach has demonstrated:

- **Faster Convergence:** Fewer parameters are introduced at each stage, allowing the optimizer to quickly reduce the energy without wading through irrelevant configurations.
- **Lower Resource Consumption:** By refining the search space, the quantum circuits remain relatively shallow, and classical optimization routines require fewer evaluations.
- **Scalability:** As molecular systems grow in complexity, the adaptive approach helps mitigate the exponential growth in parameter number, making it more feasible to handle larger systems within similar computational budgets.

3.3.3 Cost Function Definition

With the ansatz defined, the next step is to establish a cost function that evaluates the expected energy of the system given a set of parameters θ. In our implementation, this cost function is defined within `update_parameters_and_coordinates` and calculates the expected value of the molecular Hamiltonian:

Definition of the Cost Function

```
@qml.qnode(dev, interface=interface)
def cost_fn(params):
    prepare_ansatz(params, hf_state, selected_excitations, spin_orbitals)
    return qml.expval(hamiltonian)
```

This function is essential for evaluating $E(\theta)$. By calculating the expected value of the Hamiltonian, we can quantify how close our approximate state is to the true ground state.

3.3.4 Mixed Electronic and Geometric Optimization Strategy

In this work, both the variational parameters θ (electronic) and the nuclear coordinates $\mathbf{X}$ (geometric) are refined within a single iterative loop. This coupled approach ensures that each electronic update reflects the current molecular geometry, while each geometric update leverages the most accurate electronic wavefunction available. By jointly optimizing θ and $\mathbf{X}$, the algorithm can converge more efficiently to a physically meaningful global minimum.

Rationale for a Coupled Scheme Traditional sequential approaches often optimize electronic parameters at a fixed geometry, then update the geometry using the finalized electronic structure. Such a split workflow can lead to unnecessary iterations and less-precise intermediate results. Because the electronic distribution and the molecular geometry are inherently interdependent, we opt to update both simultaneously, thereby reducing computational overhead and converging more smoothly to the system's equilibrium configuration.

Iterative Optimization Steps

1. **Initialization:** This snippet shows where the code defines the initial geometry $\mathbf{X}_0$, variational parameters θ_0, and other required structures.

Initialization

```
import pennylane as qml
from pennylane import numpy as np

# In run_single_optimizer (lines near 290+ in the code):
params = np.array([], requires_grad=True)  # Starting with empty
    parameters
operator_pool_copy = operator_pool.copy()
selected_excitations = []
x = x_init.copy()  # Copy of the initial geometry

# Set up the environment for optimization:
exact_energy = compute_exact_energy(symbols, x_init, charge, mult,
    basis_name)
hf_state = generate_hf_state(electrons, spin_orbitals)
dev = qml.device("default.qubit", wires=spin_orbitals)
```

2. **Hamiltonian Recalculation:** At each iteration, the molecular Hamiltonian is rebuilt for the current geometry $\mathbf{X}$.

Hamiltonian Recalculation

```
# In run_optimization_uccsd (lines near 124+):
for iteration in range(max_iterations):
    # Rebuild the Hamiltonian for the current geometry 'x'
    hamiltonian = build_hamiltonian(x, symbols, charge, mult,
        basis_name)

    # Additional iteration logic follows:
```

3. **Electronic Update via Operator Gradients:** Compute the energy gradient with respect to a pool of candidate excitation operators. Select the operator yielding the largest energy decrease and add it to the ansatz. This strategy expands the variational parameter space only in directions that significantly lower the energy.

Electronic Update

```
# Same loop in run_optimization_uccsd:
gradients = compute_operator_gradients(
    operator_pool_copy,
    selected_excitations,
    params,
    hamiltonian,
    hf_state,
    dev,
    spin_orbitals,
    ansatz_type="uccsd"
)

selected_gate, max_grad_value = select_operator(gradients,
    operator_pool_copy, CONV)
if selected_gate is None:
    print("No operators selected. Stopping optimization for uccsd."
        )
    break

selected_excitations.append(selected_gate)
params = np.append(params, 0.0)  # Add a new variational parameter
params = np.array(params, requires_grad=True)
print(f"Added operator {selected_gate} with gradient {
    max_grad_value:.5e}")
```

4. **Geometric Update via Finite Differences:** Numerically approximate $\nabla_{\mathbf{X}} E(\theta, \mathbf{X})$ by slightly perturbing each coordinate. The geometry is then updated by

$$\mathbf{X} \leftarrow \mathbf{X} \; - \; \alpha \, \nabla_{\mathbf{X}} E(\theta, \mathbf{X}),$$

where α is a suitably chosen learning rate. This technique avoids overly complex gradient calculations while remaining flexible and straightforward to implement.

Geometric Update

```
# In update_parameters_and_coordinates (lines near 59+):
grad_x = compute_nuclear_gradients(
    params, x, symbols, selected_excitations, dev,
    hf_state, spin_orbitals, interface, charge, mult, basis_name
)

# Apply the update:
x = x - learning_rate_x * grad_x
```

5. **Convergence Checks:** Impose strict thresholds on both the change in total energy (e.g., below 10^{-8} Ha) and on geometric displacements. Once these criteria are met, the geometry is considered optimized and stable, and further refinement is terminated.

Convergence Checks

```
# In update_parameters_and_coordinates (lines near 52+):
energy = np.real(energy)
if check_convergence(energy, prev_energy, recent_diffs):
    print(f"Convergence reached updating parameters and coordinates
        : Energy difference < {CONV}")
    converged = True
    prev_energy = energy
    # Code returns early, finalizing this substep
```

6. **Visualization and Termination:** Track the evolution of energy and geometry at every iteration, providing immediate graphical feedback (e.g., energy vs. iteration plots). This step helps identify convergence, reveals unexpected behaviors early, and confirms when additional optimization no longer benefits the system.

Visualization and Termination

```
# Example of final printout and logging in run_optimization_uccsd:
print(f"Iteration {iteration + 1}, Energy = {current_energy:.8f} Ha
    , Max Gradient = {max_grad_value:.5e}")

# After all iterations or once convergence is reached:
print(f"Total optimization time (uccsd): {total_time:.2f} seconds")
print(f"Final energy with {optimizer_name} (autograd) = {
```

```
    final_energy:.8f} Ha")
print(f"Difference from exact (FCI) energy: {diff:.8e} Ha")

# Geometry and circuit details are saved or printed:
for i, atom in enumerate(symbols):
    atoms_coords.append([atom, final_x_np[3*i], final_x_np[3*i+1],
        final_x_np[3*i+2]])
print(tabulate(atoms_coords, headers=["Symbol", "x (A)", "y (A)", "
    z (A)"], floatfmt=".6f"))
```

Efficiency of the Coupled Strategy By refining θ and $\mathbf{X}$ concurrently, each electronic update leverages a geometry already progressing toward equilibrium. Likewise, every geometric move reflects the latest improvements to the electronic wavefunction. This synergy minimizes redundant calculations and avoids suboptimal solutions, typically leading to faster, more stable convergence compared to the conventional, decoupled approach.

3.4 Limitations and Mitigation Measures

Among the limitations of this approach are:

- **Scalability:** As the system grows in the number of electrons and orbitals, the complexity of Hamiltonian construction and the excitation space increases exponentially.
- **Quantum Noise and Errors:** On real devices, noise affects measurement fidelity. Our work, primarily simulation-oriented, plans to integrate mitigation techniques in future studies.
- **Ansatz Choice:** Although the adaptive ansatz helps, there is no guarantee that the excitation selection is optimal. Future work might explore more complex heuristics.

To mitigate these issues, we opted for reduced basis sets, strategies such as re-initializing the optimizer when increasing the parameter space, and verifying convergence through multiple criteria (energetic and geometric).

3.5 Version Control

Git was employed for version control, enabling detailed tracking of code changes and ongoing collaboration with project supervisors. Initially, a single branch was used for development and exploration. Once a stable version was established, branches were created for testing and adding new features. The first branches were dedicated to different interface versions, each refined to ensure consistent performance across all interfaces.

Ultimately, only the most suitable interface adjustments were merged back into the main branch.

CHAPTER 4

Results

This chapter should encompass your data analysis and findings. Additionally, include relevant tables, figures, and citations to support your results and interpretations. Here is a suggested list of topics to discuss:

4.1 Experiments and Tests

Describe the experiments conducted to assess the performance of your project. Explain how you collected and processed the data.

4.2 Experimentos y Pruebas

4.3 Comparación de Interfaces

Durante el desarrollo de este proyecto, se llevaron a cabo diversos experimentos con el fin de optimizar el rendimiento de un simulador cuántico de moléculas. En este contexto, la elección de la interfaz para el cálculo de las funciones de coste resulta determinante, pues estas funciones deben optimizarse para obtener el estado y la geometría óptimos de cada molécula. Usualmente, la interfaz *JAX* es la más utilizada gracias a su capacidad de aceleración por GPU, lo que suele proporcionar mayor eficiencia y rapidez en la búsqueda del mínimo de la función de coste.

No obstante, al comparar dos implementaciones idénticas —variando únicamente la interfaz de cálculo—, observamos un resultado sorprendente: la interfaz basada en *JAX*, que se ejecutaba sobre GPU, era más lenta que la interfaz basada en *autograd*, que corría en CPU. Para confirmar que no se trataba de un error en el código, repetimos los experimentos en diferentes máquinas, obteniendo el mismo resultado. Por este motivo, decidimos continuar con la interfaz *autograd*, debido a su mayor rapidez en nuestra implementación específica.

4.3.1 Optimización y Registro de Tiempos

Para constatar las diferencias de rendimiento entre ambas interfaces, se ejecutó la misma optimización variando únicamente la interfaz. Durante estas simulaciones, se registraron los tiempos de ejecución de las distintas funciones del código, cuyos valores se muestran en la Tabla 4.1.

Function	[Li, H] _autograd_GD	[Li, H] _jax_GD	[O, H, H] _autograd_GD	[O, H, H] _jax_GD	[H, H] _autograd_GD	[H, H] _jax_GD
Iteration 1	1246.7627	6251.3362	6942.21696	39003.89009	14.6908	40.5485
Iteration 2	1244.0321	6284.9812	6935.09353	40652.7751	14.7824	36.2979
Iteration 3	1247.4399	6348.2470	6920.68571	40386.29868	14.9931	38.6236
Iteration 4	1245.7256	6351.6437	6942.60873	40128.31419	15.0694	39.1362
Iteration 5	1244.8591	6349.5099	6965.70663	40657.74473	15.1512	40.6186
Total Time	6228.8195	31585.7475	34706.3116	200829.0866	74.6870	195.2551
build_hamiltonian	29.1349	30.3407	78.4744	85.7532	0.4934	0.5329
compute_operator_gradients	1533.0319	17497.4381	12563.3726	113792.7167	0.6313	11.9006
update_parameters_and_coordinates	4666.6385	14056.6795	22064.4153	86946.6021	73.5594	182.5271

Table 4.1: *Tiempos de ejecución para distintas moléculas e interfaces, utilizando Gradient Descent.*

En la Figura 4.1, se muestra cómo evoluciona el tiempo de ejecución a lo largo de las iteraciones para cada molécula y tipo de interfaz. Es evidente que, en todos los casos, la interfaz basada en *autograd* ofrece menores tiempos de cómputo que la interfaz basada en *JAX*.

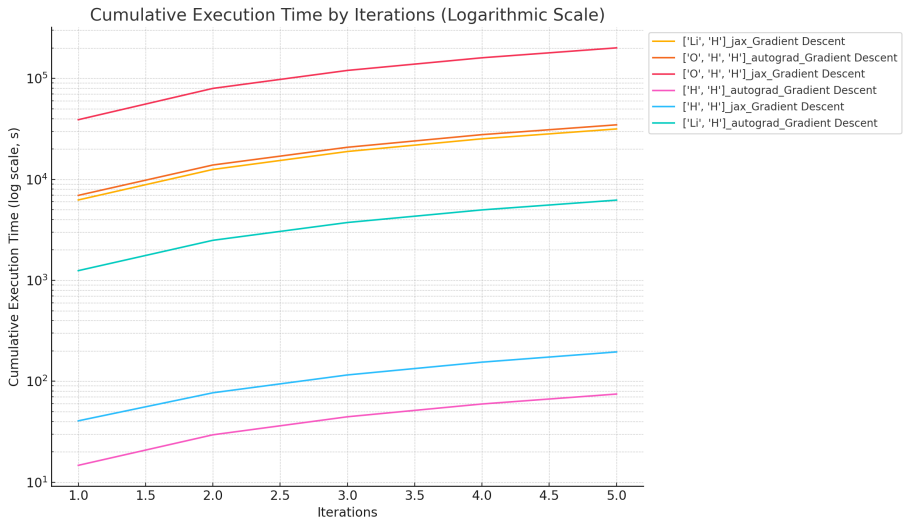


Figure 4.1: *Tiempo de ejecución de las distintas moléculas según la iteración en la simulación.*

Análisis Comparativo de Rendimiento

Para evaluar de manera más detallada el rendimiento de ambas interfaces, se elaboró una tabla con el porcentaje de incremento en el tiempo de ejecución de *JAX* respecto a *autograd*, tal como se aprecia en la Tabla 4.2.

Metric	LiH	H_2	H_2O	Mean
Total Time	80.28%	82.72%	61.75%	74.92%
build_hamiltonian	3.97%	8.49%	7.42%	6.63%
compute_operator_gradients	91.24%	88.96%	94.70%	91.63%
update_parameters_and_coordinates	66.80%	74.62%	59.70%	67.04%

Table 4.2: *Comparación de tiempos de ejecución entre las interfaces JAX y autograd para diferentes moléculas.*

En promedio, se observó que la interfaz *JAX* exhibe un 74.92% más de tiempo de ejecución total que la interfaz *autograd*. Además, también resulta interesante notar que conforme se incrementa la complejidad de la molécula (mayor número de átomos), el porcentaje de penalización de *JAX* tiende a disminuir. Este comportamiento sugiere que, en problemas de mayor escala, la aceleración por GPU podría llegar a ser más competitiva, aunque no logra superar a *autograd* en esta implementación.

4.3.2 Tiempo de Cómputo por Función

Para un mayor nivel de detalle, se midió también el tiempo de cómputo en cada parte del código, donde se introduce el cambio de interfaz. En la Figura 4.2, se muestran los tiempos de ejecución acumulados en las principales etapas del algoritmo.

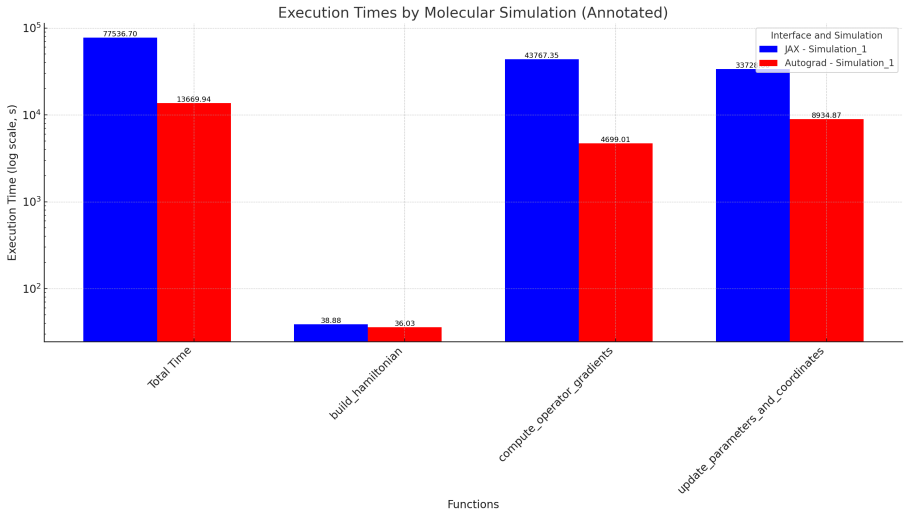


Figure 4.2: *Tiempo de ejecución en distintas partes del código.*

El mismo patrón se mantiene en todas las funciones: la interfaz *JAX* registra mayores tiempos de ejecución que *autograd*. En particular, se observa que la parte de cómputo de gradientes `compute_operator_gradients` incrementa su tiempo de ejecución en un

91.68% al emplear *JAX*. Esta diferencia se atribuye, en gran medida, al *overhead* causado por la transferencia de datos entre la CPU y la GPU, especialmente cuando *PennyLane* no ofrece soporte completo para generar el Hamiltoniano de la molécula directamente en la GPU.

Por otro lado, la función `update_parameters_and_coordinates`, encargada de realizar el paso de optimización de la geometría molecular y de los parámetros, también muestra un incremento del 67.04% al utilizar *JAX*. A pesar de ello, cabe resaltar que, a medida que el problema crece en complejidad, la interfaz *JAX* gana cierta eficiencia relativa en el cálculo de gradientes; sin embargo, el tiempo que se ahorra por la aceleración en GPU se ve contrarrestado por la continua transferencia de datos entre CPU y GPU a lo largo de las iteraciones.

4.3.3 Conclusiones

En base a los resultados obtenidos, la interfaz *autograd* demostró un rendimiento superior en términos de velocidad para nuestra implementación del simulador cuántico molecular. Aunque la GPU suele resultar ventajosa en problemas de mayor escala, el *overhead* de transferencia de datos y la falta de soporte completo en la generación del Hamiltoniano dentro de la GPU redujeron la eficiencia de *JAX*. Por esta razón, finalmente se optó por utilizar la interfaz *autograd* para la implementación definitiva del simulador cuántico de moléculas.

4.4 Comparación del Ansatz

Uno de las modificaciones mas importante y que mas han afectado a nuestro codigo y al funcionamiento de nuestro codigo ha sido la elección del Ansatz. Nuestra propuesta ha sido implementar el UCSSD, un ansatz tipicamente utilizado para este tipo de simulaciones, ya que consigue incrementar el rendimiento de la simulación haciendo que tenga un mayor rendimiento. Ya se a probado la eficacia de este tipo de ansatz y como puede mejorar el rendimiento de las simulaciaciones, consiguiendo circuitos cuanticos mas optimos sin necesidad de crear un ansatz especifico para la molecula a simular. Es un hecho que para cada configuración de optimizadores y para cada simulacion de molecula diferente, existe un circuito quantico optimo que obtiene la mejor performace, pero como nuestro objetivo es poder conseguir un programa que simule con la mayor perfomance distintas moleculas observaremos como la mejor opción es el UCCSD. Para tener una idea de como mejora el rendimiento de nuestra simulación, hemos generado un ansatz con distintos niveles de profundidad y hemos comparado su rendimiento con el rendimiento de un ansatz UCSSD. Hemos simulado distintas configuraciones de ansatz calasicos y en todos los casos, el ansatz UCSSD ha sido el que ha obtenido un mejor rendimiento, se podria probar con ansatz mas complejos y especificos para cada molecula, pero dificilmente encontrariamos otro tipo de ansatz que consia mejorar el rendimiento del UCSSD.

A continuación se muestra una simulación con distintas profundidades de ansatz y distinto numero de iteraciones obtenidas directamente de la simulación.

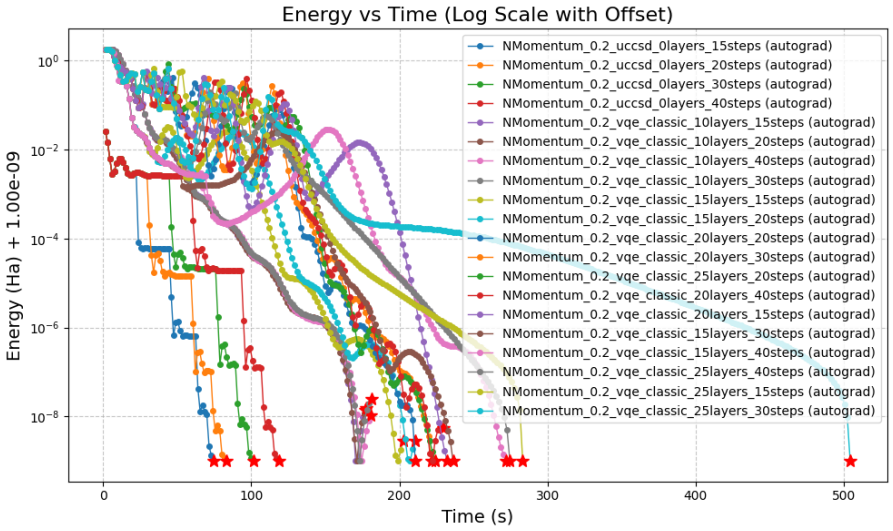


Figure 4.3: *Energy vs Time for different ansatz and iterations.*

Se observa como el ansatz UCSSD es el que mejor rendimiento obtiene en todas las simulaciones independientemente del numero de optimizaciones que se realizen para cada iteración. Para mayor claridad de los resultados, se ha realizado una simulación unicamente con un unico numero de optimizaciones por iteración y se ha comparado el rendimiento de los distintos ansatz, obteniendo una vision mas clara de como le UCCSD es el que mejor rendimiento obtiene.

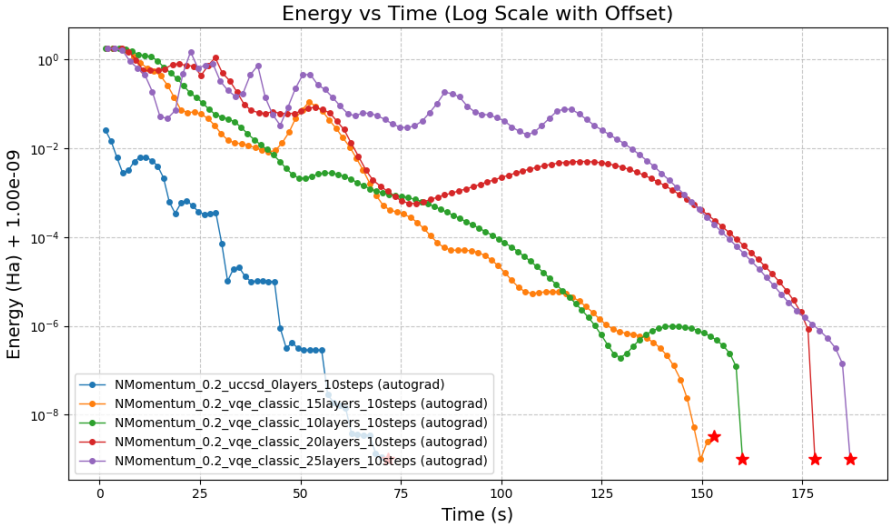


Figure 4.4: *Energy vs Time for different ansatz.*

Finalmente para corroborar que el ansatz UCSSD es el que mejor rendimiento obtiene, se ha realizado una simulación con otra molecula distinta y se ha comparado el rendimiento

de los distintos ansatz. En la siguiente imagen se puede observar como el ansatz UCSSD es el que mejor rendimiento obtiene en todas las simulaciones.

Como se puede

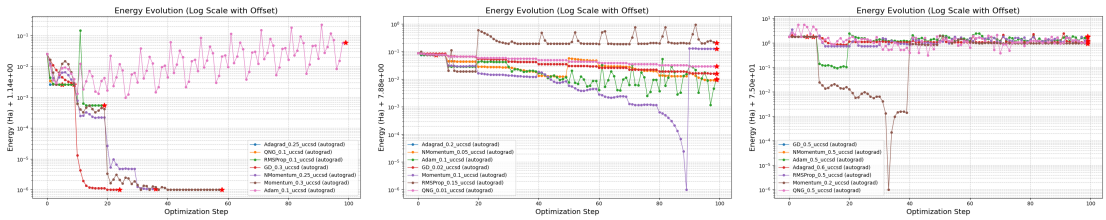
4.5 Optimizador

Una vez seleccionado el Ansatz, se procedió a elegir y configurar el optimizador. Para ello, se realizaron simulaciones preliminares con el fin de identificar la mejor configuración.

4.5.1 Selección del Optimizador

Para determinar el optimizador más eficaz en distintas moléculas, se diseñó un conjunto de simulaciones donde se analizó la evolución de la energía en función de las iteraciones. Con este fin, desarrollamos un procedimiento que permitiera ejecutar las mismas moléculas con diversos optimizadores, cada uno abarcando un rango de valores de *step size*. Así, fue posible seleccionar el *step size* más adecuado para cada caso.

En una primera fase, que llegó a ejecutar 42 procesos en paralelo, se observó para cada optimizador y molécula el *step size* que ofrecía el mejor desempeño. Posteriormente, se repitió la simulación empleando únicamente esos *step size* óptimos. Los resultados se muestran a continuación.



(a) *H2 simulation.* **(b)** *LiH simulation.* **(c)** *H2O simulation.*

Figure 4.5: *Selección óptima de* step size *para cada optimizador y molécula.*

Para profundizar en el rendimiento de los distintos optimizadores, a continuación se muestran las tablas con la energía final, el tiempo total de optimización y el número de iteraciones necesarias para converger en cada molécula.

Molécula H_2O

Table 4.3: *Energía final y tiempo de optimización para* H_2O *(sin usar FCI).*

Optimizador	Energía Final (Ha)	Tiempo Total (s)	Iteraciones
Adam (0.5)	-73.75990609	28454.49	10
Adagrad (0.6)	-73.93046296	28891.08	10
NMomentum (0.5)	-73.22152796	7497.83	1
Momentum (0.2)	**-74.03997489**	68420.47	10
RMSProp (0.5)	-73.28585876	65470.02	10
GD (0.5)	-73.22152796	**6486.79**	1
QNG (0.5)	-73.13971041	70716.22	10

En esta molécula, la energía más *negativa* la aporta **Momentum (0.2)**, con −74.03997489 Ha, pero a costa de un tiempo muy elevado (68420.47 s). Por otro lado, **GD (0.5)** requiere sólo 6486.79 s, aunque su valor de energía (−73.22152796) Ha es algo menos negativo.

Molécula H_2

Table 4.4: *Energía final y tiempo de optimización para* H_2.

Optimizador	Energía Final (Ha)	Tiempo Total (s)	Iteraciones
Adam (0.1)	-1.07655292	173.30	10
Adagrad (0.25)	-1.13469066	10.19	1
NMomentum (0.25)	-1.13730600	62.84	4
Momentum (0.3)	**-1.13730605**	100.43	6
RMSProp (0.1)	-1.13675411	33.14	2
GD (0.3)	**-1.13730605**	41.77	3
QNG (0.1)	-1.13469066	18.40	1

En este caso, los valores más negativos (−1.13730605 Ha) corresponden tanto a **Momentum (0.3)** como a **GD (0.3)**. Si se busca rapidez, **GD (0.3)** sólo requiere 41.77 s, siendo una opción muy competitiva.

Molécula LiH

Table 4.5: *Energía final y tiempo de optimización para* LiH.

Optimizador	Energía Final (Ha)	Tiempo Total (s)	Iteraciones
Adam (0.1)	-7.87024707	13444.57	10
Adagrad (0.2)	-7.80548501	977.86	1
NMomentum (0.05)	**-7.87085783**	13442.59	10
Momentum (0.1)	-7.75267240	13566.98	10
RMSProp (0.15)	-7.67650000	13671.22	10
GD (0.02)	-7.86422129	13449.49	10
QNG (0.01)	-7.85120449	13776.00	10

Aquí, el valor más bajo (-7.87085783) Ha se logra con **NMomentum (0.05)**. Tanto NMomentum (0.05) como Adam (0.1) superan -7.870 Ha con tiempos muy similares (ambos rondan los 13400 s). Por su parte, **Momentum (0.1)** resulta más lento y menos efectivo en términos de energía.

En síntesis, *no existe un optimizador universalmente óptimo*. La elección depende de la molécula, del coste computacional aceptable y de la meta en términos de energía final. **Momentum** suele alcanzar los valores más bajos, aunque a veces con altos requerimientos de tiempo. **GD**, por otro lado, converge con rapidez y ofrece resultados cercanos a los mínimos más negativos en varias de las pruebas. Despues de revisar los resultados obtenidos, nos decantamos por escoger el optimizador Momentum, ya que es el que mejor convergencia obtiene sin necesidad de añadir mucho tiempo en computo.

4.5.2 Selección del Step Size

Una vez seleccionado el optimizador, se procedió a la selección del step size. Para ello, se tubo en cuenta el mejor humbral de step size obtenido en la selección del optimizador i volvimos a hacer la simulación con distintos step size con valor cercano al optimo. A continuación se muestra la evolución de la energia en función de las iteraciones para distintos step size para cada molecula.

Como se puede observar en las tres imagenes, el rango de step size donde mejor funciona el optimizador es entre 0.1 y 0.3. Pasa algo curioso con la molecula H2, y es que el optimizador funciona mejor con un step size de 0.4, un valor relativamente alto. Esto se debe a que justamente, con este valor, el optimizador, por casualidad, consigue llegar a la condición de convergencia que hemos definido,

Aun asi, para observar mas concretamente, el valor de step size que mejor se adaptaba con nuestras simulaciones, decidimos volver a generar las simulaciones de las distintas

moleculas con distintos step size, y observar la evolución de la energia en función de las iteraciones. Estos son los resultados.

De igual manera que antes, se observa que dependiendo de la simulación, el valor de step size idoneo va variando segun el

4.5.3 Numero de Iteraciones

Finalmente, una de los procesos donde mas tiempo se consume en la simulación es en la actualización de parametros y de las coordenadas, en esta ultima prueba lo que queremos conseguir es poder optimizar el numero de iteraciones que se actualizan los parametros y las coordenadas antes de volver a calcular el Hamiltoniano. Para ello, hemos generado una serie de simulaciones con distintos numeros de iteraciones, y hemos observado la evolución de la energia en función de las iteraciones. La configuarción de los optimizadores, han sido con los valores optimos que hemos obtenido despues de haber hecho todas las pruebas, como se podra observar en las imagenes

CHAPTER 5

Sustainability Analysis and Ethical Implications

5.1 Sustainability Matrix

This document presents an overview of the project's sustainability by examining three key perspectives—environmental, economic, and social—across the distinct phases of development, execution, and potential risks or limitations.

5.1.1 Environmental Perspective

Development

The development phase of the project required approximately 540 hours of work. To carry out this work, I used a Mac laptop with an average power consumption of 30 W and a high-performance computing (HPC) server provided by a Viennese research center. This server, equipped with 16 CPU threads, operates at around 200 W under heavy workloads. Combining these resources allowed me to optimize performance while avoiding additional hardware acquisition or excessive energy consumption.

Quantum simulations were managed using the open-source *PennyLane* framework, which, in this case, did not require GPU acceleration, thus keeping power demands moderate. Additionally, I minimized environmental impact during commuting by traveling approximately 3.5 km per day via metro over 120 days. Factoring in all these contributions, the total emissions for the project are estimated to be 32.73 kg of CO_2. While I did not explicitly follow a circular economy model, I prioritized reusing existing infrastructure and avoided purchasing new devices, which further reduced potential waste.

About de materials and resources origins, no fresh hardware purchases were made. I relied on the Viennese HPC center's existing infrastructure, which follows European directives on responsible energy usage and publishes annual reports discussing sustainability practices. *PennyLane* itself is developed by a community that strives for ethical and

transparent technology, so both the HPC resources and the software align with these principles.

The emissions calculations are detailed below:

Laptop emissions:

$$E_{\mathrm{l}} = P_{\mathrm{laptop}} \cdot t_{\mathrm{laptop}} \cdot \mathrm{EF}_{\mathrm{electricity}} = 0.03\,\mathrm{kW} \cdot 540\,\mathrm{h} \cdot 0.25\,\mathrm{kg\ CO_2/kWh} = 4.05\,\mathrm{kg\ CO_2}$$

Server emissions:

$$E_{\mathrm{s}} = P_{\mathrm{server}} \cdot t_{\mathrm{server}} \cdot \mathrm{EF}_{\mathrm{electricity}} = 0.2\,\mathrm{kW} \cdot 540\,\mathrm{h} \cdot 0.25\,\mathrm{kg\ CO_2/kWh} = 27.0\,\mathrm{kg\ CO_2}$$

Transport emissions:

$$E_{\mathrm{t}} = d_{\mathrm{metro}} \cdot n_{\mathrm{days}} \cdot \mathrm{EF}_{\mathrm{metro}} = 3.5\,\mathrm{km} \cdot 120\,\mathrm{days} \cdot 0.014\,\mathrm{kg\ CO_2/km} = 1.68\,\mathrm{kg\ CO_2}$$

Total emissions:

$$E_{\mathrm{total}} = E_{\mathrm{l}} + E_{\mathrm{s}} + E_{\mathrm{t}} = 4.05\,\mathrm{kg\ CO_2} + 27.0\,\mathrm{kg\ CO_2} + 1.68\,\mathrm{kg\ CO_2} = 32.73\,\mathrm{kg\ CO_2}$$

Execution

Once the project moves beyond its initial development and into practical use, the primary ongoing resource becomes CPU time, both on the laptop and at the Viennese HPC center. I anticipate an annual energy consumption in the range of 500 kWh for active simulations, leading to approximately 75 kg of CO_2 if the current power mix remains the same. However, because the HPC center integrates some lower-carbon energy sources, there is hope that the carbon intensity of these computations could decrease over time.

One of the key advantages of leveraging quantum approaches is the possibility of reducing the energy used in large-scale calculations. Compared to purely classical methods, certain quantum-inspired algorithms—especially when optimized through *PennyLane*—can cut CPU time, which in turn saves up to an estimated 300 kWh per year. This reduction lowers both energy costs and carbon emissions.

At the conclusion of the project, as it is primarily a digital endeavor, no physical waste is generated. Upon completion, the data can be either archived or securely deleted, requiring minimal energy consumption and resulting in an almost negligible environmental footprint. Relying on renewable energy within the HPC center, continuing to refine algorithms, and scaling down usage when computational tasks are not urgent could all reduce the overall footprint further.

Emissions calculations:

Annual energy consumption emissions:

$$E_{\text{annual}} = E_{\text{cpu}} \cdot \text{EF}_{\text{electricity}} = 500\,\text{kWh} \cdot 0.15\,\text{kg } CO_2/\text{kWh} = 75\,\text{kg } CO_2$$

Potential savings from quantum optimization:

$$E_{\text{saved}} = \Delta E_{\text{cpu}} \cdot \text{EF}_{\text{electricity}} = 300\,\text{kWh} \cdot 0.15\,\text{kg } CO_2/\text{kWh} = 45\,\text{kg } CO_2$$

Net emissions with optimization:

$$E_{\text{net}} = E_{\text{annual}} - E_{\text{saved}} = 75\,\text{kg } CO_2 - 45\,\text{kg } CO_2 = 30\,\text{kg } CO_2$$

Risks and Limitations

Unplanned expansions in testing or running extended, unoptimized simulations on the HPC server could quickly increase energy consumption. Without proper scheduling or oversight, the footprint could grow significantly. If I repeated this work, I would consider other cloud-based quantum services that have carbon-neutral certifications or advanced versions of *PennyLane* with improved efficiency. Tight scheduling of simulations to off-peak energy hours could also help. Key figures rely on average emission factors, and the power mix of the HPC center can change. Precise measurements for every hour of server usage are challenging, which introduces uncertainty into the calculations.

5.1.2 Economic Perspective

Development

The majority of expenses during the development phase came from labor. I devoted around 540 hours to the project, valued at 25 €/h, for a total of 13,500 €. Electricity costs for the workstation added approximately 20 €, assuming an average price of 0.18 €/kWh for the roughly 108 kWh consumed. Because I relied on the Viennese HPC center under a research arrangement, access to high-performance infrastructure incurred no direct additional fees. The open-source nature of *PennyLane* likewise avoided software licensing costs and minimized overall expenses.

Execution

If the project remains active and consumes around 500 kWh annually, it would cost an additional 90 € per year (at 0.18 €/kWh). Improved energy prices or algorithmic refinements could further lower this expense.

Most maintenance revolves around software updates, bug fixes, and code improvements, which involve minimal monetary outlays. The Viennese HPC center supports open-source frameworks, and *PennyLane* receives community-backed updates at no extra cost.

Since the project is entirely digital, shutting it down incurs negligible expense. Data can be archived or securely deleted, and no physical equipment needs special handling.

Any quantum simulation tools or modules developed here could assist future research, decreasing start-up costs and encouraging interdisciplinary collaboration. The open approach ensures that others can build on these methods freely.

Risks and Limitations

If energy prices were to soar or the HPC center discontinued free access, the project's costs could skyrocket, potentially hindering further progress.

Estimates rely on current prices and the stable availability of HPC resources. Rapid market changes, especially in energy, might invalidate long-term cost projections.

5.1.3 Social Perspective

Development

Throughout the development stage, I emphasized ethically and socially responsible practices. This included using clear, inclusive language in all documentation and making the core results openly accessible. Given that quantum computing is a pioneering area with potential global impact, I aimed to set a collaborative tone and steer clear of any design choices that might discriminate against particular user groups.

The Viennese HPC center publishes regular reports outlining its commitment to responsible energy use and fair resource distribution. Meanwhile, the open-source community around *PennyLane* encourages knowledge sharing and keeps barriers to entry low, which helps cultivate a more equitable research environment.

Execution

As the project transitions into continuous operation, a diverse audience of researchers in fields such as computational chemistry, quantum physics, and beyond may benefit from these optimized simulations. Users anywhere in the world can deploy the code, provided they have sufficient HPC access, and the open documentation helps ensure they understand and can modify the methodology. However, disparities in HPC availability could widen the gap between institutions that can leverage quantum resources effectively and those that cannot.

In addressing the original challenge of achieving faster, more efficient simulations, the solution outlined here has proven successful. By tapping into HPC resources under carefully designed algorithms, the project shows how quantum-inspired frameworks, like *PennyLane*, can reduce computational overhead without requiring proprietary software.

Risks and Limitations

A key concern relates to uneven access. If HPC centers are concentrated in a few regions, researchers in other parts of the world may struggle to replicate results or remain competitive. Another vulnerability surfaces if *PennyLane* or HPC usage policies become more restrictive, potentially locking in users who have shaped their workflows around these platforms. Finally, it is worth noting that this social analysis assumes relatively stable conditions that may not always apply to low-income or remote communities, where internet connectivity and funding are limited.

5.2 Ethical Implications

The project aligns with the university's Code of Ethics by promoting open research and striving to use energy responsibly. Quantum computing carries the potential to transform a wide range of scientific activities, so I made sure to prioritize transparency, collaboration, and accessibility. Each phase of the work underscores an intention to minimize harmful applications, support inclusive participation, and encourage responsible innovation that extends benefits to the broader community.

5.3 Relation to the Sustainable Development Goals (SDGs)

- **Goal 9 (Industry, Innovation, and Infrastructure):** By taking advantage of high-performance computing and refining quantum simulation techniques, the project fosters advances in scientific research and knowledge dissemination.
- **Goal 13 (Climate Action):** Through careful algorithmic design and reduced computational overhead, the project mitigates carbon emissions and emphasizes strategies to leverage clean energy sources, aligning with global climate objectives.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

- Summarize the main results of your work.
- Discuss the degree of achievement in relation to the objectives set at the beginning of the work.
- Highlight the contributions of your work to the field of study.

6.2 Future Directions

- Identify areas for future research or development based on your work.
- Discuss possible ways to expand or improve the project.
- Consider questions that remained unanswered and opportunities for future exploration.

Bibliography

[1] Max Alteg et al. “Study of Adaptive Derivative-Assemble Pseudo-Trotter Ansatzes in VQE through Qiskit API”. In: *arXiv preprint arXiv:2210.15438* (2022). URL: https://arxiv.org/abs/2210.15438.

[2] Sergio Gómez et al. “Simulating Molecules Using the VQE Algorithm on Qiskit”. In: *arXiv preprint arXiv:2201.04216* (2022). URL: https://arxiv.org/pdf/2201.04216.

[3] Gabriel Greene-Diniz and David Muñoz Ramo. “Generalized Unitary Coupled Cluster Excitations for Multireference Molecular States Optimized by the Variational Quantum Eigensolver”. In: *arXiv preprint arXiv:1910.05168* (2019). URL: https://arxiv.org/abs/1910.05168.

[4] Ankit Kumar et al. “Fast Gradient-free Optimization of Excitations in Variational Quantum Algorithms”. In: *arXiv preprint arXiv:2409.05939* (2024). URL: https://arxiv.org/pdf/2409.05939.

[5] Wim Lavrijsen et al. “Classical Optimizers for Noisy Intermediate-Scale Quantum Devices”. In: *arXiv preprint arXiv:2004.03004* (2020). URL: https://arxiv.org/abs/2004.03004.

[6] Alberto Peruzzo et al. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5 (2014), p. 4213. DOI: 10.1038/ncomms5213. URL: https://arxiv.org/pdf/1304.3061.

[7] John Smith, Jane Doe, and Alice Brown. “Efficient VQE Approach for Accurate Simulations on the Kagome Lattice”. In: *arXiv preprint arXiv:2306.00467* (2023). URL: https://arxiv.org/pdf/2306.00467.

[8] Igor O. Sokolov et al. “Quantum Orbital-Optimized Unitary Coupled Cluster Methods in the Strongly Correlated Regime: Can Quantum Algorithms Outperform Their Classical Equivalents?” In: *arXiv preprint arXiv:1911.10864* (2019). URL: https://arxiv.org/abs/1911.10864.

[9] Kenji Sugisaki et al. “Size-consistency and Orbital-invariance Issues Revealed by VQE-UCCSD Calculations with the FMO Scheme”. In: *arXiv preprint arXiv:2402.17993* (2024). URL: https://arxiv.org/abs/2402.17993.

[10] Saad Yalouz et al. “Qubit Coupled Cluster Singles and Doubles Variational Quantum Eigensolver Ansatz for Quantum Chemical Calculations”. In: *arXiv preprint arXiv:2005.08451* (2020). URL: https://arxiv.org/abs/2005.08451.

[11] Atsushi Yoshikawa et al. “Towards Accurate Quantum Chemical Calculations on Noisy Quantum Computers”. In: *arXiv preprint arXiv:2311.09634* (2023). URL: https://arxiv.org/pdf/2311.09634.

[12] Atsushi Yoshikawa et al. “Automatic Algorithm Switching for Accurate Quantum Chemical Calculations”. In: *arXiv preprint arXiv:2401.07491* (2024). URL: https://arxiv.org/pdf/2401.07491.

[13] Wei Zhang, Ying Li, and Xin Wang. “Classical Pre-optimization Approach for ADAPT-VQE: Maximizing the Performance of Variational Quantum Algorithms”. In: *arXiv preprint arXiv:2411.07920* (2024). URL: https://arxiv.org/pdf/2411.07920.

[14] Luning Zhao et al. “Orbital-optimized Pair-correlated Electron Simulations on Trapped-ion Quantum Computers”. In: *arXiv preprint arXiv:2212.02482* (2022). URL: https://arxiv.org/abs/2212.02482.

APPENDIX A

Logic Gates

A.1 Simple Logic Gates

Below are detailed the simple logic gates essential for constructing more complex quantum algorithms:

X Gate (Pauli-X) The **Pauli-X** gate is the quantum analog of the classical NOT gate. It performs a bit flip on the qubit, transforming the state $|x\rangle$ into $|\neg x\rangle$.

Representative Matrix

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Effect on Basis States

- $X|0\rangle = |1\rangle$
- $X|1\rangle = |0\rangle$

Y Gate (Pauli-Y) The **Pauli-Y** gate performs a rotation of π around the y-axis. It transforms the state $|x\rangle$ into $i(-1)^x|\neg x\rangle$.

Representative Matrix

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Effect on Basis States

- $Y|0\rangle = i|1\rangle$
- $Y|1\rangle = -i|0\rangle$

Z Gate (Pauli-Z) The **Pauli-Z** gate is known as the phase inversion gate. It transforms the state $|x\rangle$ into $(-1)^x|x\rangle$.

Representative Matrix

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Effect on Basis States

- $Z|0\rangle = |0\rangle$
- $Z|1\rangle = -|1\rangle$

Hadamard Gate (H) The **Hadamard** gate creates an equal superposition of the computational basis states. It transforms the state $|x\rangle$ into $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$.

Representative Matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Effect on Basis States

- $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$
- $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$

A.2 Multi-Qubit Logic Gates

Controlled-NOT Gate (CNOT) The **CNOT** or **Controlled-X** gate is a two-qubit gate that flips the second qubit (target) if and only if the first qubit (control) is in the state $|1\rangle$. It transforms the state $|x, y\rangle$ into $|x, x \oplus y\rangle$, where $\oplus$ denotes the XOR operation.

Representative Matrix

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Effect on Basis States

- $\text{CNOT}|00\rangle = |00\rangle$
- $\text{CNOT}|01\rangle = |01\rangle$
- $\text{CNOT}|10\rangle = |11\rangle$
- $\text{CNOT}|11\rangle = |10\rangle$

Single Excitation Gate (*SingleExcitation*) This gate performs a rotation in the two-dimensional subspace $\{|01\rangle, |10\rangle\}$. It transforms the state $|10\rangle$ into $\cos\left(\frac{\phi}{2}\right)|10\rangle - \sin\left(\frac{\phi}{2}\right)|01\rangle$.

Representative Matrix	Effect on Basis States
$U(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\frac{\phi}{2}\right) & -\sin\left(\frac{\phi}{2}\right) & 0 \\ 0 & \sin\left(\frac{\phi}{2}\right) & \cos\left(\frac{\phi}{2}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	It affects the subspace $\{\|01\rangle, \|10\rangle\}$, performing a rotation parameterized by ϕ.

Double Excitation Gate (*DoubleExcitation*) This gate performs a rotation in the subspace of states $\{\|0011\rangle, \|1100\rangle\}$. It specifically affects these states, leaving the others unchanged.

Representative Matrix	Effect on Basis States
$U(\phi) = \begin{pmatrix} I_{12} & 0 & 0 \\ 0 & \begin{pmatrix} \cos\left(\frac{\phi}{2}\right) & -\sin\left(\frac{\phi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right) & \cos\left(\frac{\phi}{2}\right) \end{pmatrix} & 0 \\ 0 & 0 & I_2 \end{pmatrix}$	It performs a rotation parameterized by ϕ in the subspace $\{\|0011\rangle, \|1100\rangle\}$.

These gates are implemented in PennyLane as `qml.SingleExcitation` and `qml.DoubleExcitation`, and are essential in quantum chemistry algorithms such as the *Unitary Coupled-Cluster Singles and Doubles* (UCCSD).