On the possibility of classical client
secure delegated computation

Alexandru Cojocaru

S1575638

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2016
Abstract

As quantum technologies are showing a growing progress, we will be soon facing a physical implementation of a quantum computer. It is well-known the fact that the quantum model can surpass the computational efficiency of any classical machine. Therefore, we consider the problem of delegating a hard computation to a quantum computer, such that the input of the computation remains private.

We analyse from a complexity theoretic point of view what classes of problems can be solved in this encryption scheme. In this way, we determine to what extent we can securely take advantage of the power of a quantum server.

We indicate that there is a strong relation between how hard is to encrypt a problem and how hard is to solve it. For this reason, we further investigate what classes of problems admit this type of encryption.

Moreover, we examine a quantum protocol which also performs this secure delegated task and which achieves additional security properties. Finally, we inspect the possibility of creating a classical encryption scheme which inherits the security conditions imposed by this quantum protocol.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Alexandru Cojocaru
s1575638)
Table of Contents

1. Introduction .......................................................................................................................... 1
2. Quantum Computing .............................................................................................................. 5
   2.1. Quantum Operators ........................................................................................................ 7
   2.2. Quantum circuit model ................................................................................................... 8
   2.3. Quantum Measurements ............................................................................................... 10
   2.4. Measurement Based Quantum Computing ................................................................. 11
3. Complexity Theory ................................................................................................................ 13
   3.1. Advice Turing Machines ............................................................................................... 15
       3.1.1. \( P/poly \) ................................................................................................................ 16
       3.1.2. \( P/log \) ................................................................................................................... 18
   3.2. Oracle Turing Machines ............................................................................................... 20
       3.2.1. Examples of Oracle Turing Machines .................................................................. 22
4. Generalised Encryption Scheme .......................................................................................... 23
5. Universal Blind Quantum Computing ................................................................................ 37
6. Classical Universal Blind Quantum Computing ............................................................... 42
7. Simon Problem ...................................................................................................................... 45
   7.1. Quantum Algorithm ...................................................................................................... 46
   7.2. Classical no-go Result ................................................................................................ 48
       7.2.1. \( NP \)-Impossibility ............................................................................................... 48
       7.2.2. \( NP/poly \)-Impossibility ...................................................................................... 52
8. Forrelation Problem ............................................................................................................. 58
   8.1. Quantum Algorithm ...................................................................................................... 58
   8.2. \( k \)-Forrelation ........................................................................................................... 59
   8.3. Real Forrelation .......................................................................................................... 62
       8.3.1. Distribution Correlation ....................................................................................... 67
9. Conclusions .......................................................................................................................... 69

Bibliography ............................................................................................................................ 71
1. Introduction

The evolution of traditional computers as described by Moore’s law is challenged by a series of technological limitations. Since the power of these machines is proportional to the number of transistors they have, the trend has been the construction of smaller and more efficient components. For this reason, we will be soon facing computer constituents of size no bigger than an atom [1]. The main issue that emerges is that at this scale we cannot use anymore the laws of classical physics to describe the behaviour of these devices. Instead, we need to use the theories which govern the microscopic world [2, 3]. Thus, at this point quantum computing steps in.

Quantum Computing arose from the idea of modelling any natural phenomena using the laws of physics. The quantum mechanics principles allow us to describe any process of the nature [1, 4]. Therefore, in order to develop more powerful computing devices, scientists decided to harness quantum properties of atoms, which would provide the basis for the memory and processor of the new machine [5].

Bernstein and Vazirani [4, 5] were the first to reach the remarkable conclusion that quantum computers can solve certain problems much faster than the best classical computational model, the probabilistic Turing Machine and that no possible evolution of classical computers could compare to the power brought by the quantum paradigm [6]. This model of computation based on quantum mechanics has first attained outstanding practical results when Peter Shor and Lov Grover developed quantum algorithms for problems such as searching an element in an unsorted database or factoring large numbers [7], which outclassed the efficiency of the best known classical solutions.

The advantages of quantum computing were also emphasized by Isaac Chuang and Michael Nielsen [1] who grappled with the question: “Why this struggle to obtain full control of quantum systems?”. They argue that the main reason lies in the fact that the use of quantum theory for information processing tasks leads to an impressive speedup, which is out of reach for standard traditional machines [1, 3]. Moreover, even if this outstanding improvement in speed would not be applicable for any type of problem, the breakthrough achieved through exploiting quantum processes would allow us to gain a better understanding of the Universe [3, 5] and even uncover unknown physical phenomena. In this sense, Seth Lloyd has developed a quantum algorithm which can simulate certain quantum physical systems [8] and which could therefore play an important role in areas such as high energy physics, chemistry or cosmology [6, 8].

However, perhaps the most significant impact brought by quantum computing is revealed in the field of cryptography. Firstly, current classical public key cryptography is based on computationally infeasible problems [9]. This fact illustrates their vulnerability against a quantum computer. Because a quantum computer is able to efficiently solve hard problems such as factorization, or discrete logarithm [4, 10], it could easily break the most popular cryptographic algorithms: RSA, Diffie-Hellman, ElGamal, etc.

On the other hand, quantum information theory also provides some very promising solutions for cryptographic tasks [11]. The most trivial example is the ability to generate pure random numbers. This would guarantee that we can construct keys which secure the privacy of data, in such a way that it would be impossible for an attacker to estimate their values [3, 12]. From this property was born
the concept of quantum cryptography [11, 12]. Specifically, quantum cryptography refers to a key distribution protocol which provides information theoretic security. This means that they would not only be secure against both classical and quantum adversaries, but against any adversaries. The reason is that the security is based on the fundamental theorems of quantum physics [13] instead of difficult mathematical computations. To be precise, the keys are generated by measuring certain properties of quantum particles and by using the “Uncertainty Principle” [12] we can derive that any malicious eavesdropper is going to be detected.

Our project is related to an important component of cryptography, namely encryption, and in fact, we will be focusing on the following problem:

Suppose a client wants to determine the result of a computation on some input, but lacks the necessary resources to perform the computation himself. He has access to a server which has unbounded power and can solve the problem and send the outcome back to the client. However, the main issue is that the client wants to ensure the privacy of his data, so he needs to find a way to hide the input of the computation from the server. Moreover, as we assumed the server has limitless computational capabilities, the encryption process must not be based on solving hard mathematical problems, because then, the server could crack the cipher and get the plaintext input.

The problem described is known as secure delegated computation [14]. Considering that an encryption solution to this problem is meant to be information-theoretically secure [9] (not even an unbounded powerful attacker could decipher it), this type of protocol would play a decisive role in the advancement of cloud computing. Additionally, the invulnerability assertion makes this model of encryption a perfect candidate for a cryptographic primitive of a strong, reliable cryptosystem [4, 9].

Now, taking into account the prospective quantum speedup in solving hard problems, we analyse the secure delegated computation task in the situation where we use a quantum computer in the role of the powerful server [15] completing the computation for the client.

In this thesis we are concerned with an encryption framework called Generalised Encryption Scheme (GES) [16]. This model describes the class of encryption protocols where the classical client derives the result of a difficult computation using the server’s ability to efficiently solve computationally infeasible problems, while at the same time protecting the privacy of his input. Therefore, all the deduced results in this paper do not refer to a specific encryption protocol, but to an entire family of encryption schemes which solve the secure delegated computation task.

Our target is to study what types of problems could be solved by a client in the Generalised Encryption Scheme scenario. In other words, we examine the relation between solving a problem and the complexity of encrypting it using this framework. Thus, besides the practical motivation of securely using the resources of a quantum computer to obtain the outcome of a hard computation, the Generalised Encryption Scheme also leads to important results in the field of Complexity Theory [17].

In Section 4 we will review a series of essential theorems proved by Martin Abadi [16] regarding the implications of encrypting a problem, which emphasize the idea that the harder a problem is, the more difficult it is to encrypt without revealing anything about the input. The fundamental result obtained by Abadi shows that there is a strong connection between encryption and nonuniform complexity [17, 18]. More precisely, we have that any encryptable function can be evaluated by a nondeterministic algorithm receiving an external polynomial advice.
The main contribution of our project is the investigation, based on the aforementioned result, whether problems which are efficiently computed by a quantum machine [19] (BQP problems) admit a Generalised Encryption Scheme.

But why are we interested if we can resolve problems from this particular complexity class?

First of all, BQP is the most important and well-known quantum complexity class. It refers to the decision problems which can be solved with high probability by a quantum computer in polynomial time [1, 19]. Because quantum computing is inherently probabilistic, we can view the class BQP as the quantum analogue of the tractable classical classes P or BPP [17]. Secondly, there are not many known precise relations between quantum and classical complexity. Abadi’s results [16] tell us that we can define an encryption scheme for any problem solvable in polynomial time and at the same time that there can be no such scheme for the “hardest” problems solvable by a nondeterministic algorithm. However, because there is no acknowledged complexity connection between the classes NP and BQP [20], we do not know whether there exists an encryption scheme for every BQP problem.

What we want to show is the following no-go result: no BQP – Hard problem can be encrypted using the Generalised Encryption Scheme. Considering Abadi’s central theorem, proving this statement is equivalent to showing that the class BQP cannot be included in the class NP/poly (NP/poly [18, 20] refers to problems solved by a nondeterministic algorithm using an external polynomial advice).

Of course, proving such a statement is at least as difficult as proving that P is not contained in NP [17, 21], so instead, we give strong evidence for this extremely important complexity theory result by relativizing the relation between the 2 classes [22]. This means that we indicate the separation between the 2 complexity classes with respect to an oracle (the machines from the 2 classes are allowed to make queries to an oracle - which is a fixed language).

In Section 7 we describe a candidate problem we found for the oracle separation, named Simon’s Problem [23]. We start by proving that the problem can be solved by a BQP algorithm.

Then, using a diagonalization approach [20] we show that we can build an oracle O such that Simon’s Problem cannot be solved by a nondeterministic polynomial time algorithm which has access to O.

The second part consists of indicating that the no-go result holds even when the algorithm receives an extra polynomial size advice [24].

For this proof we use an innovative technique. We build a matrix where each line refers to a possible advice received by a nondeterministic machine, each column refers to a possible input of the problem and the cell of the matrix on position (i, j) specifies if the machine receiving advice i accepts or not the input j. An advice is considered good if the machine receiving that advice decides correctly any input. Using this method, we intend to show that we can construct an oracle, such that for every input, a fraction of the total number of advices are not good, so we can eliminate them. Consequently, when we reach the last input we can conclude that no possible polynomial advice can help a nondeterministic machine to always decide correctly, which ends our proof.

In Section 8 we study another candidate problem for the separation between the quantum and the classical complexity class, called the Forrelation Problem [25]. This is an even more interesting problem as it belongs to the class BQP – Complete. Using a series of polynomial reductions between different problems, we end at a problem which is easier to solve than Forrelation called Distribution
Correlation. For Distribution Correlation [25] we can follow the same proof we used for Simon’s problem, which implies that we can draw the same no-go conclusion for the Forrelation problem.

Our second main contribution is related to a quantum protocol named Universal Blind Quantum Computing (UBQC) [26] described in Section 5. This protocol has the same aim as the Generalised Encryption Scheme, which is, allowing a client to solve a hard problem by communicating with a quantum server. The main differences from GES are that the client is no longer entirely classical, but must have minimal quantum abilities and the problem itself must be hidden from the server [26, 27].

Starting from this protocol we investigate the existence of a classical version of UBQC (CUBQC), namely a protocol where the computation is also encrypted, but where the client is entirely classical and the interaction between the client and the quantum server is classical too. Moreover, CUBQC must satisfy the correctness and security conditions imposed by the quantum protocol UBQC.

The motivation behind this new protocol can be illustrated in the following example. Suppose we have a standard computer and would like to use a quantum server to compute for us the longest path in a graph \(G\). Then, this encryption protocol would guarantee us that we would get the correct answer without the quantum computer inferring the input \(G\), or even the fact that he was actually performing a “graph longest path” task.

The novelty regarding this topic consists of proving a connection between CUBQC and GES.

Specifically, in Section 6 we show that we can represent CUBQC as an instance of the GES framework, which meets the conditions set by UBQC. This proof implies that any result regarding what class of problems can or cannot be solved in the GES scenario also applies to the CUBQC protocol. Thus, if GES cannot solve BQP-Hard problems we have the same no-go result for CUBQC.

Because our major proofs and results are correlated with the field of Complexity Theory, we will give in Section 3 an extensive description of the necessary Complexity Theory background. We will put accent on the two models of computation which are most relevant to our project: Oracle Turing Machines [20, 22] and Advice Turing Machines [24, 28]. We are going to outline examples of problems and well-known theorems regarding these two classes and also illustrate some proof techniques useful to our contributions.

Section 2 presents an overview of the principles which lie at the basis of Quantum Computation.

The subchapters presenting the quantum operators, gates and measurements [29] are necessary for the understanding of the proof that a particular problem belongs to the quantum class \(BQP\). On the other hand, the MQBC subsection [6, 29] is not required for perceiving our results, but is useful in understanding how the UBQC quantum protocol works.
2. Quantum Computing

The central unit of quantum information and computation is the qubit [2, 3, 5], the quantum analogue of a classical bit. To physically implement qubits many different systems can be used including atoms, ions, nuclear spins and photons [1]. The classical bits can take 2 possible values, 0 or 1. Qubits analogous expression are the states $|0\rangle$ and $|1\rangle$. However, the crucial difference is that a qubit can also act as if it is in the 0 and 1 state at the same time [3, 4]. Specifically, the qubit state can be described as a linear combination of $|0\rangle$ and $|1\rangle$. We can get a better picture of this statement by considering the physical representation of a qubit: for a particle, its spin state can be aligned up (corresponding to state $|1\rangle$), down (corresponding to state $|0\rangle$) or be arbitrarily aligned, in between these 2 states (a linear combination of the up and down states) [1, 4]. Therefore, any qubit $|\psi\rangle$ can be written as a superposition of $|0\rangle$ and $|1\rangle$: $|\psi\rangle = a|0\rangle + b|1\rangle$, where $a, b \in \mathbb{C}$ are also known as amplitudes [2].

In fact, $|\psi\rangle$ is a vector in a complex Hilbert Space $\mathcal{H}$ of dimension 2, while the $|0\rangle$ and $|1\rangle$ states are vectors representing the coordinate axes of this complex space. For this reason, we say that $(|0\rangle, |1\rangle)$ constitute a computational basis [1, 3] for any qubit.

Given a classical bit we can decide if it is 0 or 1 by checking its value. On the other hand, for its quantum counterpart we cannot determine the pair of amplitudes $(a, b)$ associated with its state [2, 28]. This is because if we measure $|\psi\rangle$ then we always obtain either the value 0 or 1. Basically, a measurement would destroy the quantum state $|\psi\rangle$ and collapse it to either the state $|0\rangle$ or $|1\rangle$. Moreover, the result of the measurement is nondeterministic, we get the result 0 with probability $|a|^2$ and the result 1 with probability $|b|^2$. Since we are talking about probabilities, the 2 values should sum up to 1, and we have that $|a|^2 + |b|^2 = 1$. When we have this property for $|\psi\rangle$, we say this quantum state is a pure state [5]. In this way, we know that the norm of the vector state $|\psi\rangle$ is equal to 1 ($|\psi\rangle$ is a unit vector of the $\mathcal{H}$ space).

This main difference between the standard state of a classical bit and the “unusual” superposition of a quantum bit can be explained through the next experiment. Imagine the bit as an ordinary coin. The value of the bit represents the result of tossing that coin: heads $\rightarrow$ 0 or tails $\rightarrow$ 1. On the other hand, the qubit can also reveal the status of the coin while it is in the air. At this moment, the coin is neither heads nor tails and is rather in a continuum process alternating between the 2 values, described as a superposition of the heads and tails states (for instance $|\phi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$). Then, after the coin has landed, but before we actually see it, we know that with probability $(\frac{1}{\sqrt{2}})^2$ the coin is heads and with probability $(\frac{1}{\sqrt{2}})^2$ is tails. Finally, when we look at the coin, equivalent to making the measurement, we observe the result: either heads (state $|0\rangle$) or tails (state $|1\rangle$).

For a better understanding of the qubit concept we can also view it from a geometric perspective [3]. Let $|\psi\rangle \in \mathcal{H}$ be a quantum pure state: $|\psi\rangle = a|0\rangle + b|1\rangle$ with $|a|^2 + |b|^2 = 1$. Then, we can express $|\psi\rangle$ as:

$$|\psi\rangle = \cos \frac{\theta}{2} \cdot |0\rangle + e^{i\varphi} \cdot \sin \frac{\theta}{2} \cdot |1\rangle,$$

where $\theta \in [0, \pi], \varphi \in [0, 2\pi]$. 


The Hilbert Space $\mathcal{H}$ can be represented as a 2-sphere (Figure 1) and each pure state such as $|\psi\rangle$ is a unit vector situated on the boundary of this 2-sphere, whose north and south pole correspond to the computational basis states $|0\rangle$ and $|1\rangle$.

![2-sphere representation of Hilbert Space. Source [29]](image)

An essential question we need to address is related to the amount of information which can be stored in a qubit [13]. We might be tempted to think that since the sphere contains an unbounded number of unit vectors, each associated with a different pair $(\theta_i, \phi_i)$, we could use as storage space all the bits required for the unbounded binary representation of $\theta_i$ or $\phi_i$. Unfortunately, as shown before, the measurement makes the state $|\psi\rangle$ collapse to $|0\rangle$ or $|1\rangle$, so we just get only one bit of information[1]. Consequently, in order to have access to more information, we need to consider systems consisting of several qubits.

Such a composite system is defined as the tensor product [4, 26] between individual smaller quantum systems. Essentially, the tensor product is an operation which allows 2 different vector spaces to join together and form a bigger vector space. For instance, if we have the vector space $A$ with dimension $m$ and the vector space $B$ with dimension $n$, then the tensor product between $A$ and $B$, notated as $A \otimes B$, is a vector space of dimension $m \cdot n$. A quantum state from $A \otimes B$ is just a superposition of states $|a_i\rangle \otimes |b_j\rangle$ (which is usually written $|a_i,b_j\rangle$), where $|a_i\rangle$ is a vector from $A$ and $|b_j\rangle$ is a vector from $B$. Therefore, if we have the orthonormal basis $\{|i_1\rangle, |i_2\rangle, ..., |i_m\rangle\}$ for $A$ and the orthonormal basis $\{|j_1\rangle, |j_2\rangle, ..., |j_n\rangle\}$ for $B$, then the set $\{|i_1\rangle \otimes |j_1\rangle, |i_1\rangle \otimes |j_2\rangle, ..., |i_m\rangle \otimes |j_n\rangle\}$ forms an orthonormal basis for $A \otimes B$.

Take the case of the vector space composed of quantum states containing 2 qubits. This vector space can be written as $A \otimes B$, where $A = B = 2$-dim $\mathcal{H}$. As a result, any 2-qubit quantum state $|\psi\rangle$ can be written as a linear combination of the states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$: $|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$. Exactly as in the 1-qubit scenario, after the measurement $|\psi\rangle$ will collapse to the state $|ij\rangle$ with probability $a_{ij}, i, j \in \{0,1\}$.

An important state containing 2 qubits is $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, called the Bell state [3, 5], which introduces the concept of entanglement [2]. This notion refers to a quantum state consisting of multiple qubits, which cannot be expressed as a tensor product between individual systems, but only as an entire ensemble. In fact, we can prove that the Bell’s state is entangled, by showing there does not exist any single-qubit states $|x\rangle = a|0\rangle + b|1\rangle$ and $|y\rangle = c|0\rangle + d|1\rangle$, such that $|\Phi\rangle = |x\rangle \otimes |y\rangle$.

The fundamental property of entangled qubits is that measurements performed on them appear to be correlated as indicated in the following scenario.
Suppose we draw apart the 2 qubits of the entangled quantum state $|\Phi\rangle$ in 2 locations extremely far from each other. If we measure the first qubit of $|\Phi\rangle$, with probability $\frac{1}{2}$ we get the outcome 0 and the state after the measurement becomes $|\Phi\rangle = |00\rangle$ and with probability $\frac{1}{2}$ we obtain the result 1 and the state after the measurement becomes $|\Phi\rangle = |11\rangle$. After this, if we measure the second qubit we would normally expect the result of this measurement to also be completely random. However, it turns out that the outcome of the measurement on the second qubit is equal to the measurement result we got for the first qubit. The correlation is also visible if we try different kinds of measurements [4, 8] on the 2 qubits: there are still connections between the outputs of the measurements on the first and second qubit.

2.1. Quantum Operators

Starting from a composed quantum state, we need to determine a way to manipulate its constituent qubits in order to perform certain computations [6]. This thing is achieved using operators.

Essentially, an operator $A$ is a function from a vector space $V$ of dimension $n$ to a vector space $W$ of dimension $m$ [29]. We can view $A$ as a matrix of $m$ lines and $n$ columns. Then, given an orthogonal basis $(|v_1\rangle, |v_2\rangle, ..., |v_n\rangle)$ for $V$ and an orthogonal basis $(|w_1\rangle, |w_2\rangle, ..., |w_m\rangle)$ for $W$, the operator $A$ acts in the following way:

$$A|v_i\rangle = \sum_{j=1}^{m} A_{i,j} |w_j\rangle \quad \forall \ i \in \{1, ..., n\}$$

One important operator from any vector space $V$ to itself is the identity vector $I$, which applied on any quantum state $|v\rangle$ behaves like this: $I \cdot |v\rangle = |v\rangle$.

Another well-known operator is the outer product [1]. Consider 2 quantum states $|v\rangle \in V$ and $|w\rangle \in W$. Then, the outer product between $|v\rangle$ and $|w\rangle$, notated as $|w\rangle\langle v|$ (where $\langle v|$ is the dual\(^2\) vector of $|v\rangle$) is an operator mapping vectors from $V$ to vectors from $W$:

$$\langle w|v\rangle|z\rangle = |w\rangle \cdot \langle v|z\rangle = \langle v|z\rangle \cdot |w\rangle \in W \ \forall \ |z\rangle \in V$$

as $\langle v|z\rangle$ is just a complex number.

For any quantum operator $A$ we define an operator named the adjoint, $A^+$, which satisfies the property that for any 2 vectors $|a\rangle$ and $|b\rangle$ from a vector space $V$ we have that:

- the inner product between $|v\rangle$ and $A|w\rangle$, is equal to the inner product between $A^+|v\rangle$ and $|w\rangle$, $(\langle v|, A|w\rangle) = (A^+|v\rangle, |w\rangle)$.

The operator $A$ is self-adjoint if it satisfies the condition $A = A^+$.

An essential subset of self-adjoint operators refers to projectors [1, 4], which are mostly used in the context of state measurements. Such an operator $P$, projects a vector $v \in V$ to a vector $w \in W$, where $W$ is a subspace of $V$. Additionally, $P$ satisfies the property that: $P^2 = P$. Then, given the computational basis $(|1\rangle, |2\rangle, ..., |m\rangle)$ for the vector space $W$, the projector $P : V \rightarrow W$ is defined as:

$$P = \sum_{i=1}^{m} |i\rangle\langle i|.$$  

\(^1\)The inner product of 2 vectors $|v\rangle$ and $|w\rangle$ returns a complex number written as $(\langle v|, |w\rangle)$ or as $(|v|w\rangle)$. The dual of a vector $|v\rangle \in V$ is a function from $V$ to $\mathbb{C}$ defined as: $\langle v|$ = $(|v|, \cdot )$ which generates the inner product between $|v\rangle$ and any vector from $V$. 

7
For example, when $V$ is the 2-dimensional Hilbert space and $P = |1\rangle\langle 1|$, for every $|\psi\rangle \in V$, $|\psi\rangle = a|0\rangle + b|1\rangle$ we have:

$$P|\psi\rangle = (|1\rangle\langle 1|)(a|0\rangle + b|1\rangle) = b|1\rangle.$$ Therefore, $P$ is projecting any vector from $V$ to the subspace generated by the vector state $|1\rangle$.

A unitary operator is a function $U : V \to V$ such that $U^+U = I$. One important property of this type of operator is that the application of a unitary on two vectors does not affect their inner product [3]:

Consider 2 vectors from $V$, $|v_1\rangle$ and $|v_2\rangle$. We compute the inner product between $U|v_1\rangle$ and $U|v_2\rangle$:

$$(U|v_1\rangle, U|v_2\rangle) = (U|v_1\rangle)^+ (U|v_2\rangle) = \langle v_1 | U^+U | v_2 \rangle = \langle v_1 | I | v_2 \rangle = \langle v_1 | v_2 \rangle = (|v_1\rangle, |v_2\rangle)$$

This characteristic of unitary operators tells us that if we have an orthonormal basis for $V$, $B = (|v_1\rangle, |v_2\rangle, ..., |v_k\rangle)$, then the set $B' = (|v_1'\rangle, |v_2'\rangle, ..., |v_k'\rangle)$ obtained as $|v_i'\rangle = U|v_i\rangle \forall i \in \{1, ..., k\}$, is also an orthonormal basis for $V$, because the inner product between any 2 vectors $|v_i'\rangle, |v_j'\rangle$ of $B'$ is equal to the inner product of the correspondent vectors from $B$, $|v_i\rangle$ and $|v_j\rangle$.

### 2.2. Quantum circuit model

The quantum circuit model [6] defines a mechanism to implement any possible quantum computation.

In the classical paradigm we have the boolean circuit model which allows us to determine the value of any boolean function $f$ by using a set of gates. These gates acting on a small number of bits, are applied sequentially in such a way that the resulting circuit can simulate the behaviour of $f$.

Similarly, in the quantum case, we have the analogous quantum gates used to construct quantum circuits [1, 5]. Every quantum gate can be described by a unitary operator $U$. This fact reveals an important difference between classical and quantum gates: quantum gates are reversible, while their classical counterparts are not. This property refers to the ability to rebuild the input after the application of an operator on it. It is easy to observe that from the result of an OR-gate applied on 2 bits, we cannot recover the values of the input bits. On the other hand, using the fact that every unitary operator is invertible: $U^{-1} = U^+$, we are able to reconstruct any input by just applying the inverse of that operator.

The most common quantum gates receive as input 1 or 2 qubits. Because a quantum state is a vector in a complex Hilbert Space, we can represent any quantum gate applied on it as a matrix. For instance, a quantum gate acting on one qubit is a $2 \times 2$ matrix and in general, a quantum gate acting on $n$ qubits is written as a $2^n \times 2^n$ matrix. This also brings an important remark, namely that the number of qubits given as input to a quantum gate must be equal to the number of qubits representing the output [29].

We will now enumerate some of the most common quantum gates which we are going to use in this project [1, 8]:

- The **Hadamard gate** $H$ operates on single qubits and maps the $|0\rangle$ state to the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and the $|1\rangle$ state to $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Its matrix form is

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

while in a quantum circuit it is represented as: $\boxed{H}$

- The **X gate** is the analogue of the classical **NOT** gate, mapping $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. 


It is represented as the matrix $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

- The $Y$ gate applies to a qubit a $\pi$-rotation around the $Y$-axis in the $\mathcal{H}$ space, mapping $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. It is represented as the matrix $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$.

- The $Z$ gate applies to a qubit a $\pi$-rotation around the $Z$-axis in the $\mathcal{H}$ space, mapping $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $-|1\rangle$. It is represented as the matrix $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

- The Phase Shift gate, $R_\theta$, affects the phase of a qubit state, mapping $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $e^{i\theta}|1\rangle$. It is represented as the matrix $R_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$.

- The Controlled-NOT gate, $CNOT$, is applied on 2 qubits and has the following result: $CNOT (|x\rangle \otimes |y\rangle) = |x\rangle \otimes ((x + y) \mod 2)$. Its matrix representation is:

  $$
  CNOT = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  $$

  while its quantum circuit representation is:

  ![CNOT Circuit](image)

- The $\text{SWAP}$ gate switches 2 qubit states: $\text{SWAP} (|x\rangle \otimes |y\rangle) = |y\rangle \otimes |x\rangle$. Its matrix form is:

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

- The $\text{CSIGN}$ gate (or Controlled-Z) acts on 2 qubits in the following way:

  $$
  \text{CSIGN} (|x\rangle \otimes |y\rangle) = (-1)^{x \cdot y} (|x\rangle \otimes |y\rangle). \text{ It is represented as: } \text{CSIGN} = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & -1
  \end{bmatrix};
  $$

- The $\text{CCSIGN}$ (controlled-controlled sign) operates on 3 qubits:

  $$
  \text{CCSIGN} (|x\rangle \otimes |y\rangle \otimes |z\rangle) = (-1)^{x \cdot y \cdot z} (|x\rangle \otimes |y\rangle \otimes |z\rangle]).
  $$

Now, what we would like to achieve using the quantum circuit model, is to be able to compute any possible function. This property of a computational model, called universality, is accomplished using universal sets of gates [6]. A universal set of gates guarantees us that using a subset of quantum gates chosen from this set we are able to approximate any unitary operator.

Because the total number of quantum operators is infinite we cannot hope to precisely replicate the behaviour of any operator using only a finite set of quantum gates, but instead, we can reach a good approximation of them [7, 10]. Such an example of finite universal sets is: \{CNOT, H, R_\pi/2\}.

Alternatively, if we would use an infinite universal set of gates, then we could obtain the exact output retrieved by any unitary operator. For example, the set containing the quantum gate $CNOT$ along with any unitary operator acting on 1 qubit, or the set consisting of the gates $X$, $CNOT$ and $R_\theta$ for any possible angle $\theta$, are both infinite universal sets, able to simulate any possible computation.

After we have designed the quantum circuit which implements a specific computation, we take an input consisting of a number of qubits and sequentially apply the gates according to the scheme of the circuit [3, 29]. After this stage, we need to get the actual classical outcome of the computation by measuring (observing) the final qubits.

We will next give a more detailed description of the quantum measurement process.
2.3. Quantum Measurements

So far, we have only discussed about measuring a qubit in the computational basis \{ |0\rangle, |1\rangle \} i.e. giving outcome either \{0\} or \{1\}. In fact, a measurement can be made in any basis \( B = \{ |b_0\rangle, |b_1\rangle, ..., |b_{k-1}\rangle \} \), which is a set of \( k \) linearly independent vectors and \( k \) is the dimension of the Hilbert space of our states \[6, 13\]. Any state \( |\psi\rangle \) can be written as a linear combination of vectors from \( B \), \( |\psi\rangle = c_0|b_0\rangle + c_1|b_1\rangle + \cdots + c_{k-1}|b_{k-1}\rangle \). As a result, a measurement made in the basis \( B \) would yield as result one of these vectors \( |b_i\rangle \) with the corresponding probability \( c_i^2 \).

For example, take the quantum state \( |\psi\rangle = a|0\rangle + b|1\rangle \). Then, by measuring in the computational basis we obtain that \( |\psi\rangle \) is in state \( |0\rangle \) with probability \( |a|^2 \) and in state \( |1\rangle \) with probability \( |b|^2 \). But, suppose we choose to measure \( |\psi\rangle \) in the basis \( B' = \{ |+\rangle, |-\rangle \} \) where \( |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \) and \( |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \). We can rewrite \( |\psi\rangle \) as \( |\psi\rangle = \frac{a+b}{\sqrt{2}}|+\rangle + \frac{a-b}{\sqrt{2}}|-\rangle \). Therefore, after a measurement in basis \( B' \), we obtain that that \( |\psi\rangle \) is in state \( |+\rangle \) with probability \( \frac{|a+b|^2}{2} \) and in state \( |-\rangle \) with probability \( \frac{|a-b|^2}{2} \).

Now, consider a general quantum state \( |\phi\rangle = \sum_{i=0}^{m-1} a_i|i\rangle \). By measuring this state we get one of the \( m \) possible outputs \( \{0, 1, ..., m-1\} \), each having a probability \( P(i) \).

For every possible measurement result we can associate an operator \( M_i \), where \( M_i \) is defined on the vector space of \( |\phi\rangle \). Then, the probability of outcome \( i \) is:

\[ P(i) = \langle \phi | M_i^\dagger M_i |\phi\rangle \]

The measurement operators \[1\] must satisfy the property that: \( \sum_{i=0}^{m-1} M_i^\dagger M_i = 1 \). In this way, we are assured that the sum of probabilities of all results is equal to 1:

\[ \sum_{i=0}^{m-1} P(i) = \sum_{i=0}^{m-1} \langle \phi | M_i^\dagger M_i |\phi\rangle = \langle \phi | \left( \sum_{i=0}^{m-1} M_i^\dagger M_i \right) |\phi\rangle = \langle \phi |\phi\rangle = 1 \]

After the measurement is performed, the original quantum state \( |\phi\rangle \) is also modified depending on the outcome of the measurement \[2, 29\]. Therefore, supposing the outcome was \( j \), the state \( |\phi\rangle \) will now be equal to \( |\phi'\rangle = \frac{1}{\sqrt{P(j)}} M_j |\phi\rangle \).

**Example.** Take \( |\phi\rangle \), a unit norm vector from the 2-dim \( \mathcal{H} \) defined as: \( |\phi\rangle = a_0|0\rangle + a_1|1\rangle \), such that \( |a_0|^2 + |a_1|^2 = 1 \).

Then, for the outcome measurement 0 we define the operator \( M_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \) and similarly, for outcome measurement 1 we define the operator \( M_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \).

We observe that both \( M_0 \) and \( M_1 \) are self-adjoint projectors: \( M_0 = M_0^\dagger = M_0^2 \) and \( M_1 = M_1^\dagger = M_1^2 \).

Consequently, we can compute the probabilities for the 2 possible outcomes:

\[ P(0) = \langle \phi | M_0^\dagger M_0 |\phi\rangle = (\langle 0|a_0^* + \langle 1|a_1^* \rangle M_0 a_0|0\rangle + a_1|1\rangle) = a_0^* \cdot a_0 = |a_0|^2 \]

\[ P(1) = \langle \phi | M_1^\dagger M_1 |\phi\rangle = (\langle 0|a_0^* + \langle 1|a_1^* \rangle M_1 a_0|0\rangle + a_1|1\rangle) = a_1^* \cdot a_1 = |a_1|^2 \]

---

\(^{22}\) \( \bar{x} \) is the complex conjugate of \( x \); \( |x| \) is the modulus of \( x \).
And the state after the measurement becomes:

$$|\phi\rangle \rightarrow |\phi'\rangle = \frac{1}{\sqrt{P(0)}} M_0 |\phi\rangle = \frac{1}{|a_0|} \cdot (a_0 |0\rangle),$$

if the measurement result is 0, or

$$|\phi\rangle \rightarrow |\phi'\rangle = \frac{1}{\sqrt{P(1)}} M_1 |\phi\rangle = \frac{1}{|a_1|} \cdot (a_1 |1\rangle),$$

if the measurement result is 1.

This was an example of a projective measurement [10] where all operators $M_i$ are projectors. For this type of measurement, we observe that $M_i$ satisfy the probability condition specified above:

$$\sum_{i=0}^{m-1} M_i^+ M_i = \sum_{i=0}^{m-1} M_i = 1.$$

As we have seen, the new state after the measurement depends on the measurement result. Hence, in the time between the measurement is performed and the time we actually observe its result, the quantum state is known to be in a superposition of $m$ states, each associated with a probability $P(i)$. At that point, we can conclude that $|\phi\rangle$ is in the $\frac{1}{\sqrt{P(j)}} M_j |\phi\rangle$ state, with probability $P(j)$.

In a quantum circuit, the symbol used for the measurement of a qubit is: 

![Measurement Symbol](image)

2.4. Measurement Based Quantum Computing

*Measurement Based Quantum Computing (MBQC)* is a model for quantum computing, equivalent to the circuit model [1, 4, 29]. While in the quantum circuit paradigm we applied a sequence of operators and then performed a measurement to obtain the classical result of the computation, in MBQC we execute a sequence of measurements to reach the desired result.

The starting point of a computation in the MBQC model is a *graph state*, which is an entangled state containing a large number of qubits [6]. This graph state must be generic to allow for a wide range of computations to be completed from this pattern. Then, we perform measurements at proper basis on the qubits of this state in order to derive a certain computation. In this way, the initial entanglement of the graph state is destroyed by the measurements.

Given a graph $G$ with $V$ its set of vertices and $E$ its set of edges, we build the graph state $|G\rangle$ in the following manner. Every vertex of $G$ is associated with a qubit in the $|+\rangle$ state and every edge of $G$ is associated with a pair of entangled qubits, to which we apply the Control-Z operator. Consequently:

$$|G\rangle = \prod_{(i,j) \in E} (C - Z |ij\rangle) \cdot |+\rangle^{|V|}.$$

The qubits in the graph are either measured in the $\{|0\rangle, |1\rangle\}$ or in the $\{|+\rangle, |-\rangle\}$ basis (where $|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\theta} |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - e^{i\theta} |1\rangle)$).

Measurements in the base $\{|+\rangle, |-\rangle\}$ are used to implement computations. On the other hand, we might need to *reduce* the graph state. For instance, if we want to compute an operator acting on a single qubit, we would require the graph to be in a 1-dimensional space. Removing qubits from the initial graph state in order to match the appropriate design pattern for our computation, is achieved through measurements in the $\{|0\rangle, |1\rangle\}$ basis [29].

Now, we will focus on how does this model allow us to complete certain computations using $\{|+\rangle, |-\rangle\}$ measurements.  

11
For every single-qubit measurement, we decide upon a measurement angle $\theta$ based on the result of the previous measurements. Even though the outcomes of measurements are probabilistic we can modify the angles to make the final result of the computation deterministic $[1, 6]$.

For example, if we want to compute the operator $J(\theta) = HR_\theta$, we can define the following MBQC pattern:

$\begin{array}{c|c}
|\phi\rangle & |+\rangle \\
1 & 2
\end{array}$

The graph consists of 2 nodes, where the first one represents the input for the computation, $|\phi\rangle = a|0\rangle + b|1\rangle$.

Therefore, the graph state becomes:

$|G\rangle = C - Z(|\phi\rangle|+) = C - Z((a|0\rangle + b|1\rangle)|+) = a|0\rangle|+\rangle + b|1\rangle|\rangle$

Now, we measure the first qubit in the $\{|+, -\rangle\}$ basis. The measurement projectors are:

$P_{+\theta} = |+\theta\rangle\langle +\theta| = \frac{1}{2}(|0\rangle\langle 0| + e^{-i\theta}|0\rangle\langle 1| + e^{i\theta}|1\rangle\langle 0| + |1\rangle\langle 1|)$ and

$P_{-\theta} = |-\theta\rangle\langle -\theta| = \frac{1}{2}(|0\rangle\langle 0| - e^{-i\theta}|0\rangle\langle 1| - e^{i\theta}|1\rangle\langle 0| + |1\rangle\langle 1|)$

Then, if the outcome $r$ of the measurement equals to 1, our state becomes:

$P_{+\theta}|G\rangle = |+\theta\rangle(a|+\rangle + be^{-i\theta}|\rangle)$

But, we observe that the result of applying $J(-\theta)$ on the input state $|\phi\rangle$ is: $J(-\theta)|\phi\rangle = (a|+\rangle + be^{-i\theta}|\rangle)$, which is exactly what we have obtained at the second qubit of our state.

Therefore, if the outcome of the measurement is 1, the result of this MBQC pattern is $J(-\theta)$.

Otherwise, if the result $r$ of the measurement is 0, our state becomes:

$P_{-\theta}|G\rangle = |-\theta\rangle(a|+\rangle - be^{i\theta}|\rangle)$

Using the relation $XJ(-\theta)|\phi\rangle = a|+\rangle - be^{i\theta}|\rangle$, we observe that in this case the outcome of the MBQC pattern is the application of the $XJ(-\theta)$ gate on the second qubit.

Thus, to obtain the desired result we should apply the operator $X^r$. If the measurement result is 0, we get $X^0J(-\theta) = J(-\theta)$ and if the result is 1, we obtain $X \cdot XJ(-\theta) = J(-\theta)$.

So, this means that using this MBQC pattern we can simulate the operator $J(\theta)$ for any angle $\theta$. Given the fact that every operator $U$ acting on a single qubit can be written as $U = J(0) \cdot J(\theta_1) \cdot J(\theta_2) \cdot J(\theta_3)$, we obtain that the MBQC model allows us to simulate any single-qubit operator $U$.

Moreover, we can also simulate the Controlled-Z gate by just setting an edge between 2 input states. Now, because the set of gates $\{\text{Controlled-Z}, J(\theta)\}$ is universal, this tells us that the in MBQC we can implement any possible computation $[30]$.

One thing we observed in the example with the operator $J(\theta)$ is that depending on the result of a measurement, we might need to apply a correction operator (an $X$ gate in our example). Because the measurement results are not known at the moment when we define the MBQC scheme for a
computation, we will instead modify the angles for the future measurements (for instance, measure the next qubit in the rotated basis if \( r = 1 \)).

Adapting the angles of the measurements in order to overcome the randomness of the measurements outcomes, is based on the following relations [3]. Any measurement \( M_i \) in the \( \{ |+\theta\rangle, |-\theta\rangle \} \) basis adds the operator \( X^r \) (\( r \) the result of \( M_i \)) for all the following qubits and the \( Z^r \) operator for all the qubits situated in the graph at distance = 2 from qubit \( i \). Also, any measurement \( M_j \) in the \( \{ |0\rangle, |1\rangle \} \) basis adds the operator \( Z^r \) (\( r \) the result of \( M_j \)) on all the qubits which are neighbours to \( j \).

Consequently, we set for every qubit \( i \) in the graph an initial angle \( \alpha_i \) which allows us to perform a specific computation. An \( X \) operator applied on qubit \( i \) will change its initial angle \( \alpha_i \) to \( -\alpha_i \), whereas a \( Z \) operator will change \( \alpha_i \) to \( \alpha_i + \pi \). Therefore, for any qubit \( i \) in the graph we cumulate all the angle modifications resulting from the previous measurements and we obtain that the new angle is:

\[
\alpha_i' = (-1)^s \alpha_i + t\pi,
\]

where \( s = \) number of qubits where an \( X \) correction is applied, \( t = \) number of qubits where the \( Z \) correction is applied [4, 29].

### 3. Complexity Theory

After we presented an introduction to the field of Quantum Computing, we cover some necessary background in *Complexity Theory*, which is essential for the understanding of our project.

We proceed by defining the complexity classes, both classical and quantum, which are going to be encountered in our proofs [17, 20, 21].

- **\( P \)** is the class of problems which can be solved *efficiently in polynomial time* by a deterministic *Turing Machine* [17]. We say that a language \( L \) belongs to the class \( P \) if we can build a deterministic Turing Machine \( M \) running in polynomial time, such that for every \( x \in L \), \( M \) accepts \( x \) (returns 1) and for every \( x \notin L \), \( M \) rejects \( x \) (returns 0).

- **\( NP \)** is the class of problems which can be computed in *polynomial time* by a nondeterministic *Turing Machine* [20]. Every \( NP \) algorithm has 2 stages: the first in which he makes a *guess* for the result of the problem and the second where he *verifies* in polynomial time if the guess was a correct answer. A useful way to describe the behaviour of any \( NP \) algorithm is by looking at its *computational tree* [21]. The computational tree indicates how does the algorithm work, given a specific input \( x \) to the problem. The initial configuration of the Turing Machine on the input \( x \) represents the root of the tree. Every internal node corresponds to a computation performed by the algorithm and the children of the node are the possible configurations which can be obtained in one computational step. For each nonterminal node, the choice for the next computation is made in a nondeterministic way [17].
In a decision problem, the leaves of the computational tree are either accepting or rejecting states. Then, we say that a string \( s \) is accepted by the language defined in the decision problem, if in the computational tree generated for \( s \) there exists at least one path ending in an accepting state. On the other hand, a string \( s \) is rejected if all possible paths of the tree end in a reject state [33].

- **NP – Hard** problems are the problems at least as difficult to solve as the hardest problems in the NP class. We say a problem \( G \) belongs to NP – Hard if every problem from NP can be reduced efficiently in polynomial time to \( G \).
- The **NP – Complete** class contains the problems which are both in NP and in NP – Hard.
- A decision problem \( G \) belongs to the class co-NP, if the complement of \( G \), obtained by switching the accepting answers with the rejecting answers and vice versa, is in the NP class [20, 32]. In this way, we have that a string \( x \) is accepted if every path of the computational tree ends in an accepting state and is rejected if there exists one path ending in a rejecting state.


- The BPP class (Bounded-error probabilistic polynomial time) contains the problems which can be solved by probabilistic Turing Machines [16, 34] in polynomial time, with the probability of giving a wrong answer being less than \( \frac{1}{3} \). We say a language \( L \) is in the BPP class if we can construct a probabilistic Turing Machine \( M \) running in polynomial time such that:
  - If \( x \in L \), then \( M \) accepts \( x \) with probability \( p \geq \frac{2}{3} \)
  - If \( x \notin L \), then \( M \) accepts \( x \) with probability \( p \leq \frac{1}{3} \)

- The ZPP class (Zero-error probabilistic polynomial time) consists of the problems which can be solved in polynomial time by probabilistic Turing Machines which may also return the answer “Do not know” for some inputs. Specifically, a probabilistic Turing Machine \( M \) which solves a problem \( G \) from ZPP acts in the following way:
  - \( M \) gives the correct answer for an input \( x \) with probability \( p > \frac{1}{2} \)
  - \( M \) answers “Do not know” for an input \( x \) with probability \( p < \frac{1}{2} \).

- The BQP class (Bounded-error quantum polynomial time) is the quantum equivalent of the BPP class. BQP consists of the problems which can be solved in polynomial time by Quantum Turing Machines with the probability of giving a wrong answer being less than \( \frac{1}{3} \) [19]. We can prove that a decision problem \( G \) belongs to BQP by showing that there exists a quantum circuit of polynomial size (as defined in the quantum circuit model section) which can solve \( G \) with bounded-error [19, 30].

Now, we need to focus on 2 particular categories of problems: problems which can be solved by Turing Machines which receive an additional “help” called advice [28] and problems which can be
solved by Turing Machines which have access to an oracle [22] that can correctly answer to particular questions. We are going to give a thorough description of these two models of computation as they are directly connected with our major results. In order to prove our main statements, we need a full understanding of the behaviour of the machines which can solve these classes of problems. Hence, together with the formal definitions, we will also present examples and results regarding the two models, which we are going to use in our proofs.

3.1. Advice Turing Machines

An advice function is any function \( f: \mathbb{N} \to \Sigma^* \).

Turing Machines with an advice function \( f \) receive an additional help to solve problems, in the form of an advice string \( f(n) \). The most important property of the advice function is that this external information does not depend on the value of the input for a problem, but only on the input size [20]. That is, for a given input \( x \), the received advice is \( f(|x|) \).

**DEFINITION 1:** The class \( C/F \) of languages recognized by Turing Machines with advice is the set of languages \( A \) associated with a language \( C \) from \( C \) and an advice function \( f \) from \( F \) such that:
\[
A = \{ x \in \Sigma^* \mid \text{the pair } (x, f(|x|)) \in C, C \in C, f \in F \}.
\]

In other words, we say that there exists a function \( f \) from \( F \), which gives the necessary additional information to a machine from the class \( C \), in order to accept a more difficult language belonging to the class \( C/F \).

We will now present a series of classes of advice languages and their interpretation [17, 18, 28]:

- **EXAMPLE 1:** \( F = \log \) represents the set of functions \( f \) satisfying the property:
  \[
  \forall n \in \mathbb{N} \exists c \in \mathbb{N} \text{ such that } |f(n)| \leq c \cdot \log n.
  \]
  We define \( P/\log \) as the class of languages \( L = \{ x \mid (x, f(|x|)) \in A, A \in P, f \in \log \} \).
  Thus, for any such language \( L \in P/poly \) we can determine if a string \( x \) belongs to \( L \) by using a deterministic polynomial time algorithm with the help of an external information \( s \), where \( |s| = \log(x) \).

- **EXAMPLE 2:** \( F = \text{poly} \) represents the set of functions \( f \) satisfying the property:
  \[
  \forall n, \exists \text{ a polynomial } p \text{ such that } |f(n)| \leq p(n).
  \]
  We define \( P/poly \) as the class of languages \( L = \{ x \mid (x, f(|x|)) \in A, A \in P, f \in \text{poly} \} \).

- **EXAMPLE 3:** And similarly, \( NP/poly \) is represented by the set of languages:
  \[
  L = \{ x \mid (x, f(|x|)) \in A, A \in NP, f \in \text{poly} \}.
  \]
  If the advice is the empty string then we obtain exactly the languages from the class \( C \), therefore, we obtain that \( C \subseteq C/F \), and particularly \( P \subseteq P/poly \) or \( NP \subseteq NP/\log \).
3.1.1. \textit{P/poly}

The external information can be an extremely powerful tool allowing a machine to decide complex languages which initially could not have been decided by the machine.

This is best exemplified by the relation between bounded error probabilistic polynomial time machines (\(BPP\)) and deterministic polynomial time machines (\(P\)). As we know, every \(TM\) in \(P\) can be simulated by a probabilistic \(TM\) in \(BPP\), so \(P \subseteq BPP\) [24].

But, we can show that by receiving a polynomial advice every deterministic machine is also able to simulate a probabilistic \(TM\) from \(BPP\) [28]:

\textbf{THEOREM 1:} \(BPP \subseteq P/poly\).

\textbf{Proof.} Consider a language \(L \in BPP\), decided by a \(BPP\) machine \(M\). \(M\) receives 2 inputs: a string \(x\) and a random string \(r\). Supposing the input \(x\) is of length \(n\), then as \(M\) is a polynomial time machine, it runs on \(x\) in \(p(n)\) time, with \(p\) a polynomial. Moreover, the length of the random string \(r\) must also less or equal than \(p(n)\).

We intend to prove that we can use the random string as an advice to a deterministic polynomial time machine in order to decide \(L\), so we must show that for every possible length \(m\), there exists a random string \(r_m\) of size \(p(m)\) for which:

\[ M(x, r_m) = \begin{cases} 1, & x \in L \\ 0, & x \notin L \end{cases} \quad \forall x \text{ of length } m. \]

We define the set of \textit{wrong} random strings as the strings which cannot be the advice:

\[ \text{wrong}(x) = \{ r \in \{0, 1\}^{p(n)} \mid M(x, r) \text{ gives an incorrect answer} \}. \]

Therefore, what we need to show is that if we rule out all strings from the set \(\text{wrong}(x)\) for all \(x\), we will still have at least one string left in \(\Sigma^* - \cup_{x \in \{0, 1\}^n} \text{wrong}(x)\) and this string will allow \(M\) to always give the correct answer.

Using \textit{amplification} of \(BPP\) [21], we can obtain that the error probability of \(M\) is at most \(\frac{1}{2^m}\).

Then, this means that \(P(r \in \text{wrong}(x), r \in \{0, 1\}^{p(n)}) = \frac{1}{2^m} \cdot \frac{1}{2^{2n}} = \frac{1}{2^n}\).

Generalising for all inputs \(x\) of \(\text{length } n\) and by using \textit{Boole inequality}[34] we reach the relation:

\[ P \left( r \in \bigcup_{x \in \{0, 1\}^n} \text{wrong}(x) \right) \leq \sum_{x \in \{0, 1\}^n} P(r \in \text{wrong}(x)) = 2^n \cdot \frac{1}{2^{2n}} = \frac{1}{2^n} \]

This demonstrates that there is an exponentially small number of random strings which give the \textit{wrong} answer. Consequently, we can choose one random string \(r_c\) from the complement of the \textit{wrong} set. Then, we know that \(M(x, r_c)\) must be correct for any input \(x\) of length \(n\). Therefore, if an “adviser” gives us this string \(r_c\), we can use it as advice to always decide correctly if an input belongs or not to the language \(L\). \qed

\[ \text{Given a set of actions } \{A_1, A_2, \ldots, A_k\}, \text{ the probability of at least one } A_i \text{ taking place is less or equal than the sum of probabilities that each action takes place:} \]

\[ P(\bigcup_{i=1}^k A_i) \leq \sum_{i=1}^k P(A_i) \]
Another example of non-uniform models (where we basically have a different algorithm for every possible size of the input) refers to Boolean Circuits \([17, 20]\).

**DEFINITION 1.1:** A **Boolean Circuit** helps us compute a binary function by sequentially applying the logical gates \(\text{AND}, \text{OR}, \text{NOT}\) on a given input. The computation is represented as a directed graph with \(V\) – the set of vertices, \(E\) – set of edges and a tag function \(t : V \rightarrow \{x_1, x_2, ..., x_m, 0, 1\} \cup \{\text{NOT, AND, OR}\} \cup \{\text{output}\}\), defined as:

\[
t(v) = \begin{cases} 
\text{NOT,} & \text{in-degree}(v) = 1 \\
\text{AND} \mid \text{OR}, & \text{in-degree}(v) = 2 \\
x_1 \mid x_2 \mid ... \mid x_m \mid 0 \mid 1, & \text{in-degree}(v) = 0 \text{ (input vertices)} \\
\text{output}, & \text{in-degree}(v) = 1, \text{out-degree}(v) = 0
\end{cases}
\]

Essentially, the graph consists of \(m\) input nodes, 1 output node (which denotes the result of the computation) and internal nodes named gates which are tagged with the boolean operations \(\text{AND, OR, NOT}\) (Figure 2). We can consider that each node has a value associated with it: input nodes \(x_i\) have exactly the value \(x_i\), while for every other node, the value is obtained by applying the gate indicated by its tag to the values corresponding to its children nodes \([21]\).

![Figure 2. Example of Boolean Circuit which implements the XOR function](image)

The connection between **Boolean Circuits** and **Advice Turing Machines** can be observed in the fact that to decide a language \(L\), then for every possible size \(n\) of an input we need to define a new circuit which receives \(n\) inputs.

**DEFINITION 1.2:** We say a language \(L\) has **polynomial circuits** if there is a polynomial \(p\) and an array of circuits \(\{C_n\}\) satisfying the following properties:

1. The number of gates in the circuit \(C_m \leq p(m) \forall m\);
2. \(C_m\) receives \(m\) inputs and returns 1 if \(x \in L\) and 0 if \(x \notin L\) \(\forall x \in \{0, 1\}^m\);

There is a strong relation between polynomial circuits and machines with polynomial advice \([20]\):

**THEOREM 2:** \(L \in P/\text{poly} \Rightarrow L\) has polynomial circuits.

**Proof.** \("\Rightarrow\) Consider \(L \in P/\text{poly}\). Then, there exists a polynomial advice function \(f\) and a set \(A \in \mathcal{P}\) such that: \(x \in L\) \(\iff (x, f(|x|)) \in A\).

For every possible length \(n\) we can build a circuit \(C_m\), which has as input nodes the bits of a input \(x\) of length \(n\) and the \(|f(n)|\) bits representing the advice. This circuit just returns the pair \((x, f(|x|))\) which is given as an input to a second circuit \(C_m'\) implementing \(A\) (\(C'_m\) must exist, as \(A\) can be solved in polynomial time). Consequently, \(C_m'\) will give the output 1 if \((x, f(|x|)) \in A\) \(\iff x \in L\) and 0 otherwise \((x \notin L)\). In other words, every boolean circuit \(C_m'\) decides \(L\) for every input \(x\) of length \(m\). \(\square\)
“⇐”. Now, take a language \( L \) which has polynomial circuits. We want to build a deterministic polynomial-time machine \( M \) which decides \( L \) with the aid of a polynomial external information. We can encode each circuit \( C_m \) using the alphabet \( \{0, 1\} \) into a string \( s_m \). According to the definition of \( L \), there exists a polynomial \( p \) such that \( C_m \leq p(m) \). Then we also know that the resulting encoding of \( C_m \) can be determined in polynomial time, so \( |s_m| \leq p(m) \). In this way, we can use \( s_m \) as a polynomial advice for every input of size \( m \) to figure whether an input of length \( m \) is in \( L \) or not. Therefore, \( L \in P/poly \). □

The extra power brought by the polynomial advice to a deterministic polynomial-time machine can also be viewed in the following result [18, 24]:

**Theorem 3**: The class \( P/poly \) contains undecidable languages.

This is a remarkable fact given that a non-recursive (undecidable) language [17] is a language \( L \) for which there does not exist any \( TM \) which halts on any input \( x \) and decides correctly if \( x \) is or not in \( L \).

**Proof**: We demonstrate our claim by first showing that every unary language (language in which all the strings are of the form \( 1^k \)) is in the class \( P/poly \).

Take \( L_u \), a unary language. As every element of \( L_u \) has the form \( 1^k \), there exists a unique element of length \( n \) in \( L_u \), \( \forall n \in \mathbb{N} \). Then, we can build the advice string for a deterministic polynomial time \( TM \) \( M \) in the following way: for each possible length of the input, \( m \), we have the advice string \( s_m = 1 \), if \( 1^m \in L_u \) and \( s_m = 0 \) otherwise.

Thus, \( M \) decides correctly for every possible input if it belongs to \( L_u \) or not.

Secondly, we show that there exist unary languages which are undecidable.

We select any undecidable language from \( \{0, 1\}^* \), call it \( L_N \). Starting from \( L_N \) we construct the language:

\[
L_u' = \{1^{k(x)} \mid x \in L_N \text{ and } k(x) \text{ is the number resulting from treating the binary string } x \text{ as a number}\}
\]

Obviously, \( L_u' \) is a unary language. Moreover, since any machine which can decide \( L_u' \) can also decide \( L_N \) (there is a reduction from the language \( L_N \) to \( L_u' \)) and since \( L_N \) is non-recursive, we deduce that \( L_u' \) is also non-recursive. □

3.1.2. \( P/log \)

In the case of logarithmic advice, the external information can rather be seen as an index of an element in a set of uniformly created elements [20].

\( P/log \) class also shows a separation from the uniform \( TMs \), as the logarithmic advice helps the deterministic machine solve additional problems which initially could not be determined.

We will now present an interesting proof method [24, 30] involving the use of logarithmic advice which helped us gain intuition for our main proofs.

In this approach we will take into consideration every possible advice and then eliminate the candidates which do not lead to a correct answer. We are going to adapt this technique for our proof that a particular problem cannot be solved by any deterministic machine receiving external advice.
**THEOREM 4:** If the SAT problem can be solved by a $P/log$ algorithm, then $P = NP$.

**Proof:** Consider a Boolean formula $F$ containing $n$ boolean variables: $x_1, x_2, \ldots, x_n$.

The advice being logarithmic in the size of the input, for the formula $F$ we get an advice $s$, such that: $|s| \leq c \cdot \log n$, $c$ constant.

Because SAT $\in P/log$, there exist a language $A$ in $P$ such that $(F, s) \in A$.

Then, we show that we can determine if $F$ is satisfiable using the following deterministic polynomial time algorithm [20].

```plaintext
for every $a \in \Sigma^*$ such that $|a| \leq c \cdot n$ do // we try every potential logarithmic advice
    $F' = F$
    while $F'$ still contains variables:
        choose a variable $x_i$ from $F'$
        $F'_1 = \text{formula obtained after replacing all occurrences of } x_i \text{ with } 0 \text{ in } F'$
        $F'_2 = \text{formula obtained after replacing all occurrences of } x_i \text{ with } 1 \text{ in } F'$
        if $(F'_1, a) \in A$ then $F' = F'_1$
        else if $(F'_2, a) \in A$ then $F' = F'_2$
        else break;
    end while
    if $F' = \text{true}$ then halt and return “yes” // found a satisfying assignment
end for
halt and reject // no assignment has been found
```

Using the variable $a$ we try every possible candidate for the logarithm advice for the input $F$. Even if at some point we find that for a string $a$ we have $(F, a) \in A$, we don’t stop here as this $a$ may not be the correct advice (it might be the case that $F$ is not satisfiable and as $a$ is not the real advice for $F$ the pair $(F, a)$ is accepted by $A$). Instead, we continue assigning values to the variables in $F$ until there are no more variables in $F$. Only then we will be certain that the considered $a$ was the correct advice and that $F$ was indeed satisfiable.

In this way, if the algorithm returns “yes”, then we know for sure that $F$ must be satisfiable. Furthermore, if $F$ is satisfiable, then there exists a value for the variable $a$ for which the machine $A$ gives the correct answer.

If $F$ contains $n$ variables, then the presented algorithm runs in $O(n)$ (as the while loop is executed $2^{\log n} = n$ times).

Therefore, SAT can be solved by a polynomial-time algorithm. However, because SAT $\in NPComplete$ [17], meaning that every NP problem can be reduced to SAT, it results that $NP = P$. □

3.2. Oracle Turing Machines

**Definition 2**: An oracle $TM$ is a machine with access to an oracle which can decide if a string is in a language $O \subseteq \Sigma^*$.

In addition to the standard $TM$, an oracle $TM$ has an oracle tape [20] which can be used to query for an extra input and the machine will receive in only one computational step an answer specifying if that input belongs to $O$ or not.

Every normal $TM$ can be considered an oracle $TM$ by setting the empty set $\emptyset$ as the oracle’s language.

We can encode oracle Turing Machines in the same manner we do for standard $TM$ and by conversion to the alphabet $\Sigma = \{0, 1\}^*$, we can view these encoding as natural numbers [17]. This means that the set of oracle $TM$ is countable, so we can enumerate all these machines: $\{M_0^1, M_0^2, \ldots, M_0^n, \ldots\}$.

For any oracle machine $M_o$ we are able to describe its computational tree as presented below.

We first consider that the oracle’s language has not been set yet and we run $M_o$ on input $m$. Then, we set the configuration of $M_o$ on input $m$ as the root of the computational tree. Next, every inner node of the tree represents a query made by $M_o$ to the oracle $O$. For every such node we have 2 branches defining 2 different sets of computations which $M_o$ might perform depending on the result of the query [21]. So, if the queried string belongs to $O$, then we proceed to the computations specified in the left branch, otherwise to the computations indicated by the right branch. In the nodes where no more queries are made (the leaves of the tree), $M_o$ will take a decision, if it is an accepting state or not [22].

Now, we set the language $L$ for the oracle $O$. Consequently, we will obtain a single path $\pi$ in the computation tree starting from the root and ending in an accept or reject node. For every node $i$ the path goes to its left child if the string labelled at node $i$ is inside the language $L$ and goes to its right child otherwise.

Finally, we conclude that the initial input $m$ is in the language accepted by the oracle $TM M_o$ if this resulted path $\pi$ terminates in an accepting state.

Comparing the structures of the computational trees of an oracle $TM$ and of an nondeterministic $TM$ (described in the introduction of the $NP$ class), we can intuit the relation between a nondeterministic $TM$ and a deterministic oracle $TM$ [31]. Specifically, we can prove that we are able to simulate the behaviour of a nondeterministic $TM$ on an oracle deterministic $TM$ and also that the reversed relation is not true [30, 31].

**Theorem 5**: Any nondeterministic $TM$ can be simulated by an oracle deterministic $TM$.

**Proof.** Let $M_o$ be an oracle deterministic machine and $M'$ be a nondeterministic machine. Then, we want $M_o$ to be able to simulate $M'$ using the oracle $O$. We remind that an oracle can decide any language $L$, so we define $O$ in a rather unusual way:

$O$ will contain the encodings of all computations performed by the nondeterministic algorithm of $M'$. In this way, by querying the oracle we can check if a specific computation was indeed performed by $M'$, or even better if a specific configuration is valid for $M'$. For every nondeterministic action of $M'$, $M_o$ will use the oracle $O$ to decide between the possible following steps.

This simulation procedure can be described in the next pseudocode.
We can observe that by querying the oracle we reproduce the nondeterminism of $M'$. Moreover, if $M'$ accepts an input $x$, then this deterministic algorithm will find a path $\pi$ in the computational tree of $M'$ generated for $x$ such that $\pi$ ends in an accepting state. □

**Lemma 5.1:** There exist deterministic oracle TMs which cannot be simulated by nondeterministic standard TMs.

**Proof.** The intuition behind this statement is that, in general, oracle machines can be extremely powerful computational systems [20, 21], by defining languages difficult to decide.

This can be illustrated by the fact that a deterministic oracle machine can decide any language $L$ solved by a nondeterministic machine $M'$ [17]. For instance, we can define the language of the oracle as the language accepted by $M'$. Consequently, a deterministic $TM$ with oracle $O$ only needs to take the input for $M'$ and query $O$. Then, the answer received from the oracle is the output of $M'$. □

**Lemma 5.2:** We can define an algorithm which allows the simulation of some oracle deterministic $TM$ by standard nondeterministic machines.

**Proof.** Suppose we have a deterministic machine with oracle $O$, $M_o$ and a nondeterministic $TM$ $M'$. Then $M'$ could copy all the deterministic computations of $M_o$ until it makes a query to the oracle. At this point, $M'$ can use its nondeterminism to choose the continuation associated with a response for $M_o$ from the oracle $O$.

The main issue arises when the oracle queries the same input multiple times. Then, the nondeterministic machine might select different continuations (because it does not look at what choices it made previously), which leads to inconsistency (giving different results in response to queries for identical inputs) [22, 33].

Moreover, $M'$ does not know the language defined by the oracle $O$, so the only thing he can do is consider every possible oracle which $M_o$ might have, and accept an input $x$ if it is accepted by any of these resulting oracle TMs. □
3.2.1. Examples of Oracle Turing Machines

**Definition 3:** We define $P^O$ as the set of problems which can be solved by a deterministic polynomial-time $TM$ which has access to the oracle $O$. Similarly, we have that $NP^O$ is the set of problems solved by a nondeterministic polynomial-time $TM$ with access to the oracle $O$.

Whenever $O$ is a language which cannot be decided by $M$, the oracle adds more computational power to $M$ [31]. For instance, $P^{SAT}$ represents the set of problems solved by a deterministic $TM$ which also has access to an oracle capable of solving the $SAT$ problem. This oracle will obviously help compute more functions. In general, we have:

**Axiom 6.1:** $C \subseteq C^O$ for any oracle $O$ and any complexity class $C$.

**Lemma 6.2:** If $O$ can be solved by a deterministic polynomial time machine we can show that: $P^O = P$.

**Proof.** We already know from **Axiom 6.1** that $P \subseteq P^O$ (1). Now, since $O \in P$, the verification algorithm used by the oracle to check if a string belongs to his language, is just a polynomial time procedure. Thus, we are able to convert every deterministic $TM$ using oracle $O$ into a different deterministic polynomial time $TM$. Consequently, we also have $P^O \subseteq P$ (2).

From (1) and (2) we obtain $P^O = P$. □

Adding an oracle allows us to identify relations between different complexity classes. Therefore, proving separations between classes of problems, with respect to a given fixed oracle represents a strong evidence that those separations would also hold in the lack of the oracle [32, 33].

**Theorem 7.** We can find an oracle $B$ such that the following crucial result holds: $P^B \neq NP^B$.

**Proof.** Given any language $B$ we build the language $L_B = \{1^n \mid \exists \text{ string } s \in B \text{ such that } |s| = n\}$.

To decide if a string $s = 1^m$ is in $L_B$ we can simply make a choice for a string $s \in \{0, 1\}^m$ and then verify with the help of the oracle if $s \in B$. Therefore, we have shown that $L_B \in NP^B$, $\forall B$.

Now, we want to build the language $B$ such that $L_B$ cannot be decided by any deterministic $TM$ with access to the oracle $B$.

The set of Turing Machines is countable, so we can list all deterministic $TM$ [17], each of them being associated with a natural number $i: M_i \in P$. We create the oracle $B$ in a series of steps, so that at step $i$ we make sure that $M_i^B$ is not able to correctly decide $L_B$.

Thus, when we reach step $i$ we already have determined if a number of strings $\{s_1, \ldots, s_k\}$ belong to $B$. We pick $m$ such that $m > |s_i| \ \forall i \in \{1, \ldots, k\}$. Then, we run $M_i$ on the input $1^m$ and we want $M_i$ to return the wrong answer.

If $M_i$ returns “yes” for $1^m$, we remove all the strings of length $m$ from the set $B$, so that we would have $1^m \notin L_B$.

If $M_i$ returns “no” for $1^m$, then we can add to $B$ a string $s'$, $|s'| = m$, such that $s'$ has not been queried before (we can find such a string because $M_i \in P$ can query for a polynomial number of strings, but we have an exponential number of available strings of length $m$). Thereby, $1^m \in L_B$.

By this construction, we made sure that $M_i$ returns the wrong answer, so we can conclude that $L_B$ cannot be in the set $P^B$ and hence $P^B \neq NP^B$. □
It is worth mentioning that in the case of a nondeterministic oracle \( TM \) \( M' \), the computational tree of \( M' \) contains two types of nodes: nodes where the machine queries the oracle and nondeterministic nodes \([20]\). After setting an oracle, we define the continuation from the query nodes and we are left with the nondeterministic states. Then, \( M' \) accepts an input \( x \) if in this resulting computational tree there exists one accepting path \([33]\).

In this project we aim to show that 2 complexity classes of problems \( BQP \) and \( NP/poly \cap coNP/poly \) are different. Because there is no known relation between \( BQP \) and \( NP \) \([30]\), we give a strong evidence for our claim by stating that they are different with respect to an oracle \( O \) \([31]\). Therefore, we define an oracle relative to which a problem is in the first class, but is not in the latter:

\[ \exists \text{ Oracle } O \text{ such that } BQP^O \not\subseteq NP/poly^O \cap coNP/poly^O \]

Moreover, for the problems we considered to indicate the separation between the 2 classes, we use an oracle \( O \), which instead of giving a yes/no answer, will return the value of a function on a given point. We can adapt the definition of an oracle to match this scenario in the following way.

The oracle will receive a string containing: an index \( i \) used to identify the desired function \( f_i \), the length of the input for \( f_i \), an input \( x \) and the index \( j \) of a bit. The oracle will return a 0/1 answer representing the \( j \)-th bit of the value of \( f_i \) on input \( x \):

\[ \langle i, n, x, j \rangle \xrightarrow{O} \text{ the } j \text{-th bit of } f_i(x), \text{ where } f_i \text{ is the } i \text{-th function } f_i : \{0, 1\}^n \rightarrow \{0, 1\}^n. \]

Now, that we have covered all the necessary background we move to the central part of our project which focuses on the Generalised Encryption Scheme and the Complexity Theory results related to it.

### 4. Generalised Encryption Scheme

Consider the following situation: a client wants to compute a hard function. Because he does not have the necessary computational resources to do it by himself, he wants to use an untrusted server which is capable of solving difficult problems. In this case, the client needs to somehow encrypt his input to the problem, such that the server cannot find out any kind of information about it.

This is the purpose of Generalised Encryption Scheme (GES) \([16]\).

**DEFINITION 4:** GES is an encryption scheme framework solving the delegated computation task \([14]\) and can be described by the following sequence of steps:

- **S1.** The client applies the encryption primitive on his input and sends the resulting cyphertext to the server.
- **S2.** The server performs the computation on the ciphered data and returns the outcome back to the client.
- **S3.** The client applies the decryption procedure and in this way he obtains the result of his desired computation.
OBS 1: A first observation is that this encryption scheme must satisfy the property that by applying the hard function on the encrypted input and then decrypting the resulting outcome, will give us the result of applying the function on the cleartext input.

Basically, this scheme would allow any user with limited computational capabilities to solve any problem by exploiting a powerful server, while protecting the privacy of his input. Moreover, this scheme guarantees information-theoretic security [9]. This means that no matter how powerful a malicious server might be and no matter what actions he might try (or even if a third party could have access to the encrypted input), the security of this protocol cannot be broken [12].

OBS 2: Achieving information-theoretic security implies that the encryption must not be based on computationally infeasible problems. If we assume that the server would have unlimited power, then he would be able to solve those problems [11] and obtain the plaintext input of the client.

Therefore, by taking into consideration its security premises, this is an extremely important problem, as a GES protocol would bring major consequences in cloud computing and in principle it can guarantee secure processes over the Internet [4, 14].

Furthermore, this scheme has significant consequences in the field of Complexity Theory [16, 17]. We will next derive relations between the complexity of solving a problem \( f \) and the complexity of encrypting \( f \) in the GES scenario. Specifically, we will analyse the implications for a function which belongs to a complexity class \( C \), to admit a GES scheme.

Below, we are going to give a precise description of the GES framework.

Suppose a client \( A \) wants to compute the value of \( f(x) \), where \( f : M \to N \) and \( x \in M \). Then, \( A \) must choose a key \( k \in K \) which will be used along the entire evolution of the protocol. The encryption primitive \( E \) takes as input \( x \) and the key \( k \) and returns the cyphertext \( y, E : M \times K \to M \). The obtained string \( y \) is then sent to the server \( B \) which applies the function \( f \) and returns \( f(y) \) to \( A \).

The decryption primitive \( D \) takes \( f(y) \), the input \( x \) and the key \( k \) and returns the desired result:

\[
D : N \times M \times K \to M \quad \text{such that} \quad D(f(y), x, k) = f(x), \quad \text{where} \quad y = E(x)
\]

OBS 3: In terms of privacy, the scheme does not necessarily need to hide everything about input \( x \). For instance, the client may only want to hide the first half of the bits of the input.

Therefore, we consider a function \( L \) which takes as input a string from \( M \) and defines how much does the scheme leak about an input \( x \) (what is the maximum allowed information about an input \( x \in M \) that can be inferred by the server \( B \)) [16]. From now on, all the results obtained for specific classes of problems solvable in the GES scenario will be made in relation to the function \( L \).

DEFINITION 5: GES is defined in a broader context as an interactive protocol in which \( A \) and \( B \) will exchange \( m \) pairs of messages in the following way:

- At the beginning, \( A \) uses a randomized key-generation procedure \( K() \) which receives \( x \) and returns a key \( k \in K \) which will be used in all the \( m \) sessions of communication;
- Then, \( A \) calls the encryption function \( E \) on the pair \((x, k)\) and sends as a first message \( a_1 = E(x, k) \). Accordingly, \( B \) responds back to \( A \) with a message \( b_1 \);
- For the next round, \( A \)'s encryption procedure will also consider \( B \)'s response \( b_1 \) and will generate the new message \( a_2 = E(x, k, [b_1]) \). Hence, at step \( i \), \( A \)'s message will be based on
all previous message received from B: $a_i = E(x, k, [b_1, b_2, ..., b_{i-1}])$, which is thereafter sent to B;

- $B$ chooses his messages, $(b_1, b_2, ..., b_m)$, in such a way that after the last round of communication, when $A$ finally applies the decryption function which takes all $m$ messages received from $B$, the result $r = D(x, k, [b_1, b_2, ..., b_m])$ is equal to $f(x)$ with probability $\frac{1}{2} + \frac{1}{|\mathcal{E}|}$, where $c$ is a constant.

We can observe that $B$’s responses $b_i$ do not even have to be elements from the codomain of $f$; they can be chosen from any distribution known by $B$, as long as they satisfy the condition mentioned above.

**OBS 4:** Considering that the client $A$ has only limited computational power we must add the restraints that the number of rounds, $m$, must be polynomial in $|x|$ and that the procedures of encryption $E(\cdot)$, decryption $D(\cdot)$ and key-generation $k(\cdot)$ must run in polynomial time. Particularly, as $k(\cdot)$ is a randomized algorithm, $k(\cdot) \in \text{ZPP}$ [20].

We will now present some examples of hard problems [7, 17] and show how they are solved using the GES protocol, namely we describe how the encryption, decryption and key-generation functions are defined for each of these problems [16].

**Example 1:** Suppose $f$ is the discrete logarithm problem (which is known to be in $NP$ and $BQP$). In this problem we are given a prime number $p$, a generator $g$ of the group $\mathbb{Z}_p$ and an element $u$ of $\mathbb{Z}_p$, $u \in \{0, 1, ..., p - 1\}$. The problem asks us to determine the unique number $e \in \mathbb{Z}_p$ such that $g^e \equiv u \mod p$.

Therefore, the input $x$ consists of the tuple $(p, g, u)$. We set the key-generation procedure to return a random number $k \in \mathbb{Z}_p$ and choose the encryption procedure to act in the following way:

$$E(p, g, u, k) = (p, g, v), \text{ where } v = (u \cdot g^k) \mod p.$$  

Then, $B$ receives $(p, g, v)$ and solves the discrete logarithm problem for this input. This means that he determines the value $e' \in \mathbb{Z}_p$ such that $g^{e'} \equiv v \mod p$.

Now, we need to find a decryption procedure which given $e'$ and $k$ returns the desired result $e$. This decryption function is $D(e', k) = (e' - k) \mod p$.

We can show that this procedure does return the correct solution by verifying that $g^e \equiv u \mod p$, where $e = (e' - k) \mod p$.

$$g^{e} = g^{(e' - k) \mod p} \equiv g^{e'} \cdot (g^k)^{-1} \mod p.$$  

But, we know that $g^{e'} \equiv v \mod p$, so we obtain:

$$g^{e} \equiv v \cdot (g^k)^{-1} \mod p = u \mod p.$$  

In this way, we have defined a GES scheme for the discrete logarithm problem which leaks the values of $p$ and $g$ (as $p$ and $g$ are sent directly in plaintext to $B$).

**Example 2:** Consider $f$ to be the primitive root problem. This problem receives as input a prime number $p$ and an integer $g$ and checks if the number $g$ is a primitive root modulo $p$. This is equivalent to verifying that for every integer $a$ coprime with $p$ there exists an integer $k$ such that $g^k \equiv a \mod p$. 

25
The key-generation procedure returns an integer $k \in \{1, \ldots, p-1\}$ such that $k$ is coprime with $p-1$.

The encryption procedure is defined as: $E((p, g), k) = (p, g')$, where $g' = g^k \mod p$.

Then, $B$ receives $(p, g')$ and returns 1 if $g'$ is a primitive root modulo $p$ and 0 otherwise. Once $A$ receives this outcome the decryption function just needs to copy this answer due to the next relation: $g'$ is primitive root mod $p$ $\iff$ $g$ is a primitive root mod $p$.

Thus, we found a GES scheme for the primitive root problem leaking $p$ and the order of $g \mod p$. □

**Example 3:** Suppose $f$ is the quadratic residuosity problem. In this problem we are given a number $n$, such that $n = p \cdot q$, with $p$ and $q$ prime numbers and $u \in \mathbb{Z}$. We need to check if there exists a number $b \in \mathbb{Z}$, such that $u \equiv b^2 \mod n$.

The key generation procedure returns a key $k$ consisting of a pair of numbers $(c, d)$, where $c \in \mathbb{Z}_n$ and $d \in \{0, 1\}$, both numbers selected at random.

The encryption primitive is constructed as: $E((n, u), k) = (n, v)$, where $v = (u \cdot c^2 \cdot (-1)^d) \mod n$.

Therefore, $B$ gets $(n, v)$ and determines whether $v$ is a quadratic residue modulo $n$ and returns to $A$ the answer $r$.

The decryption function used by $A$ returns $r$ if $d$ was 0, and $1 - r$ if $d$ was 1.

As seen, this GES scheme built for the quadratic residuosity problem leaks only the number $n$. □

We now move on to analyse the connections between the complexity class of a problem $f$ and how hard is solving $f$ using GES. The first important result is the following [16]:

**Theorem 8.** If $f$ belongs to the class ZPP, then $f$ can be solved using a GES scheme that hides everything about the input.

**Proof.** If $f \in ZPP$ we can define the following GES scheme:

We select a random number $z$ from the domain of $f$. Then, we can define the 3 primitives of the client in this way:

$K(x) = f(x); \quad E(x, k) = z; \quad D(r, k, x) = k;$

Basically, because the problem is in ZPP it can be directly computed by the client. Therefore, the key-generation procedure can be used to determine the desired result, $f(x)$. The encryption just returns a random element from the domain, making sure that the server cannot infer any information about $x$, not even its size. □

**Theorem 9.** If $f$ can be solved using a GES scheme that hides even the length of the input, then $f$ must be in the class ZPP.

**Proof.** Given a function $f$ computed in a GES scheme which hides everything about the input, including its length, we show that we can construct an algorithm in ZPP that can solve $f$.

First, we select a fixed input $z$ from the domain of $f$. 26
We know that the encryption procedure runs in polynomial time (OBS 4), therefore there exists a polynomial \( p \) such that \( \forall k \in \mathcal{K} \) valid for the input \( z \), we have that \( E(z,k) = y_k \) and \( |y_k| \leq p(|z|) \).

We define the set \( \mathcal{Y} = \{ y \in \text{Domain}(f) | |y| \leq p(|z|) \} \). Definitely, we have that \( \mathcal{Y} \) is finite and that any possible encryption of \( z \) must be in this set.

Take \( x \) any input for the function \( f \). Because the protocol also needs to hide the size of \( x \), then its encryption \( y \) must lie in \( \mathcal{Y} \). Otherwise, if \( |y| > p(|z|) \), then \( B \) could infer some information about the length of \( x \) (for instance that \( |x| \) is different from \( |z| \)).

Now, as we know that the number of possible outcomes of \( E \) is finite (less or equal than \( \text{card}(\mathcal{Y}) \)) we can perform an initial computation [21](before the actual running of the algorithm), where we store in a table \( T \) the values of \( f(y) \) for any \( y \in \mathcal{Y} \).

Then, the ZPP algorithm for solving \( f \) is the following:

```plaintext
Algorithm f(x) {
    Draw a key \( k \) from \( \mathcal{K} \); // this is the probabilistic part of the algorithm, as we might
    // select a key \( k \) which is not valid for \( x \), so there is a probability that
    // the final answer of this algorithm is "DO NOT KNOW"
    Compute \( y = E(x) \);
    Search the value of \( f(y) \) in table \( T \);
    Apply decryption \( D(f(y),k,x) = f(x) \);
}
```

These 2 results show us that if GES can hide the length of the input, then the scheme can hide everything about it, but if we do not want to leak anything about the input, then the function must be in ZPP [16, 17].

Because this privacy constraint of hiding everything about the client’s data might be unnecessary and because it prevents any hard functions from being computed in the GES scenario, we will instead consider GES schemes that leak the size of the input, \( L(x) = |x| \).

Based on the proofs for the ZPP functions, we might be tempted to believe that the difficulty of encrypting functions using GES, grows with the difficulty of solving them. However, this is not the case, as illustrated by the next example.

**Example 4.** We construct a function \( f \) in the following manner: for each possible length \( n \) of an input \( x \) we select a Turing Machine \( \mathcal{M}_n \) whose encoding is a string \( s \), such that \( |s| = |x| = n \).

Then \( f \) is defined as:

\[
f(x) = \begin{cases} 1, & \text{if } \mathcal{M}_{|x|} \text{ halts for all possible inputs} \\ 0, & \text{otherwise} \end{cases}
\]

Determining whether a TM halts on all inputs is known to be an undecidable problem [34] (there does not exist any algorithm which can solve this decision problem). By the construction of \( f \), we obtain that \( f \) is also undecidable.
However, there exists a GES scheme which could solve $f$ and which leaks only the length of the input. The encryption function picks any random number $y$ from the domain of $f$ such that $|x| = |y|$. Then, $B$ would solve the problem for the input $y$, and the corresponding answer, $f(y)$, will be equal to the outcome for our input $x$ (because $f(x) = f(y)$ for any $y$ such that $|y| = |x|$).

The presented counterexample indicates that if $f(x)$ can be written as a function of the length $|x|$ (∃ function $h$ such that $f(x) = h(|x|)$), then $f$ can be encrypted using a GES leaking $L(x) = |x|$. Therefore, in this case, no matter how hard is the function $h$, $f$ can be solved using the GES protocol. □

**OBS 5:** This conclusion reveals the link between classes of problems solvable in the GES protocol and classes of problems solvable using advice functions.

We remind that the advice function is an external information helping a machine to solve different problems, which does not depend on the value of the input, but only on its length [28]. For instance, $P/poly$ is defined as the class of languages:

$$C = \{x \mid (x, h(|x|)) \in A, A \in P \text{ and } h(n) \leq poly(n) \forall n\}.$$  

Then, we observe that for every function $f$ such that $f(x) = h(|x|)$, we have that $f$ belongs to the class $P/poly$ [24]. This is because for any input $x$, the advice could directly give the value $h(|x|)$, thus solving $f$.

Using the result of Karp [20] which says that no NP-hard problems can be solved by a $P/poly$ machine, we obtain that if $f$ only depends on the length of its inputs and $f$ is NP-hard, then $f$ cannot be solved using a GES protocol which leaks $L(x) = |x|$.

Now, we want to move to the general case for any type of function $f$ and study the relation between encrypting $f$ and computing $f$ with the aid of an external advice.

We have the following crucial result of Abadi [16]:

**MAIN THEOREM 1:** If a function $f$ can be solved using a GES scheme which leaks the size of the input, then: $f \in NP/poly \cap coNP/poly$.

**Basic case.** We begin our proof from a simpler scenario of GES:

- 1 round of communication between $A$ and $B$ in which $A$ sends $y = E(x, k)$, $B$ computes $f(y)$ and returns it to $A$ and finally, $A$ decrypts $f(y)$ and obtains $f(x)$.

We can define the following algorithm in $NP/poly$ which computes the function $f$.

For any possible length $n$ we consider the input string $0^n$ and choose a valid key $k_n$ for $0^n$.

The advice can be any kind of information which could help an algorithm solve $f$, as long as it is the same piece of data for all inputs of the same length. Therefore, for the length $n$ we define the advice as the pair $(u_n, v_n)$, where $u_n = E(0^n, k_n)$ and $v_n = f(u_n)$.
Then, the nondeterministic algorithm using this advice is the following [16]:

```plaintext
f(x) {
    from the advice function we receive the pair (u_n, v_n) where n = |x|;
    k = choice from the space of keys, \mathcal{K}, such that E(x, k) = u_n; // the nondeterministic step
    apply the decryption function D(v_n, k, x) = f(x);
}
```

The correctness of this algorithm is guaranteed by the fact that we can find a key k such that \(E(x, k) = E(0^n, k_n)\). This is due to the fact that since the encryption function leaks only the input size, then for any inputs \(x_1, x_2\) such that \(|x_1| = |x_2|\), there exist 2 keys \(k_1, k_2\) such that \(E(x_1, k_1) = E(x_2, k_2)\).

Now, we can advance to the general GES scenario. The main issue in this case is not the fact that there are multiple rounds of interaction between \(A\) and \(B\), but that the final result \(A\) obtains is \(f(x)\) only with some probability \(p = \frac{1}{2} + \frac{1}{|x|^c}\). Showing that \(f\) can be solved by an NP algorithm using advice, implies we must always get the correct answer \(f(x)\) [17].

**DEFINITION 5.1:** We define a transcript \(t\) as the set of pairs of messages exchanged between \(A\) and \(B\) during the GES protocol, \(t = [(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)]\).

The final result derived by \(A\) is based on the input \(x\), a key \(k\) (chosen by \(A\) at the beginning of the protocol) and the transcript \(t\) denoting the actual communication between \(A\) and \(B\). Therefore, this outcome is a function of \(x, k\) and \(t\), \(\text{out} = F(x, k, t)\) and from the definition of GES we know that \(\text{out} = f(x)\) with probability \(p\).

We can associate for every input \(x\), a distribution of possible transcripts \(t\). As the encryption hides everything but the length of \(x\), this probability distribution of transcripts is equal for all inputs of the same size. Therefore, for every possible length \(n\) of the input, we have a corresponding random variable \(T_n\) representing the transcript distribution [34].

Below, we are going to make some additional notations which will help us during the proof.

**DEFINITION 5.2:** We call a transcript \(t\) suitable for a given input \(x\) and a key \(k\) if there is a strictly greater than \(0\) probability that the actual communication between \(A\) and \(B\), is equal to \(t\), given \(x\) and \(k\).

**DEFINITION 5.3:** \(\mathcal{K}_{x,t}\) is the set of keys \(k_i\) such that the transcript \(t\) is suitable for \(x\) and \(k_i\). Verifying that a key belongs to this set can be done in polynomial time: we fix \(B\)’s responses exactly as they appear in transcript \(t\), then compute \(A\)’s behaviour given the key \(k\), input \(x\) and \(B\)’s responses and lastly, verify if \(A\)’s messages match the set \((a_1, \ldots, a_m)\) from \(t\).

**DEFINITION 5.4:** Thus, we say that a key \(k\) is good for an input \(x\) and a transcript \(t\), if \(k \in \mathcal{K}_{x,t}\) and the final result obtained by \(A\), \(\text{out} = F(x, k, t)\), is equal to \(f(x)\); \(\mathcal{K}_{x,t}\) is the set of good keys, while the rest of the keys, which we call wrong keys, are in the set \(\mathcal{K}_{x,t}\).

Looking back at the previous proof for the Basic case (with 1 round of communication between \(A\) and \(B\)), we notice that the advice consisted of the pair \((y, f(y))\), which can be seen as a transcript with only 2 messages. In the general case of GES, we intend to determine a set \(S = \{t_1, \ldots, t_l\}\) containing a
polynomial number of well-chosen transcripts such that $S$ will be used as the advice to a nondeterministic machine in order to compute $f(x)$.

While in the Basic case, the nondeterministic step was the choice of $k$ which led us to the same encryption $t_n$ for all inputs of size $n$, in the general GES scenario the nondeterministic computation will be represented by some choices of keys $k$ from the sets of good keys $k g x t r$, where $t_i \in S$.

Suppose the advice $S$ would consist of only one transcript $t$. Then, we must be able to find a key $k_i$ such that for any input $x$, $k_i$ is good for $(x, t)$. But there might be the case that using the same key $k_i$ for any $x$, would not always generate the same transcript $t$ [32]. Therefore, we need to consider more than just one transcript for $S$.

Now we must determine which transcripts should be chosen as advice.

The key used by the $A$ along the entire protocol is the outcome of the random key-generation procedure $k( )$ which runs in polynomial time. As $k( )$ is a randomized algorithm ($k( ) \epsilon ZPP$), it uses an additional tape consisting of an infinite row of random bits. Thus, $k( )$ can be perceived as a deterministic procedure which receives 2 inputs: $x$ and $w$, where $w$ is comprised of random bits from that extra tape [17, 20].

We examine the case when this key-generation primitive $k( )$ has only a finite number of random bits which it can access and consider this number to be polynomial in $n, r(n)$.

DEFINITION 5.5: For every choice of the polynomial $r$, we obtain a new reduced GES $G_r$ [16], created from the original GES $G$. Consider $k_r( )$ the new key generation function that uses exactly $r(n)$ random bits for every $x$ of size $n$. Then, the reduced GES $G_r$ is defined in the following way:

Given an input $x$, $k_r( )$ produces a key $k'$. Then, the bits of $k'$ will be used as a tape for the key-generation procedure of $G$, $k( )$. After $k( )$ returns a key $k$, $G_r$ just replicates the behaviour of $G$. Essentially, we do not alter the algorithm of $k( )$, we just restrict the size of the domain for $k( )$. Therefore, $G_r$ should give us the same final answer we would have obtained if we used $G$.

However, the key-generation procedure of $G k( )$, might require more than $r(n)$ random bits and in this case $G_r$ would stop and we would not have any transcript at all.

We describe the probability of this event happening as:

DEFINITION 5.6. $break_r(x, t) = P(k( ) requires more than r(n) bits | T_n = t and X = x)$.

DEFINITION 5.7. We denote $Break_r(x)$ as the expected value of $break_r(x, t)$ over the distribution of $t$ Hence, $Break_r(x)$ defines the probability that the reduced $G_r$ cannot deliver any transcript for an input $x$, if we only allow $r(|x|)$ random bits to $k( )$.

To analyse how appropriate is a transcript to be used as advice, we need to consider the following indicators:

$fail(x, t) = P(G obtains a result different from $f(x)$ | $T_n = t$ and $X = x$)

$fail_r(x, t) = P(G obtains a result different from $f(x)$ | $T_n = t$ and $X = x$ and

$k( ) requires no more than r(n) bits$)

$= P(G_r obtains a result different from $f(x)$ | $T_n = t$ and $X = x$)
We observe that \( fail_r(x, t) = 1 \) if \( G \) returns the correct answer only when \( k() \) needs more than \( r(n) \) bits.

We use the conditional probability relation [34]:

\[
P(U \mid V, W) = \frac{P(U, W, V)}{P(W, V)} = \frac{P(U, W \mid V)P(V)}{P(W \mid V)P(V)} = \frac{P(U, W \mid V)}{P(W \mid V)}
\]

If we make the substitutions: \( U = "G \) obtains the wrong result", \( V = "T_n = t and X = x" \) and \( W = "k() \) uses at most \( r(n) \) bits", we obtain that: \( \text{fail}_r(x, t) = P(U \mid V, W) \), \( \text{break}_r(x, t) = 1 - P(W \mid V) \) and \( \text{fail}(x, t) = P(U \mid V) \). Therefore, we have:

\[
\text{fail}_r(x, t) = \frac{P(U, W \mid V)}{1 - \text{break}_r(x, t)} \leq \frac{\text{fail}(x, t)}{1 - \text{break}_r(x, t)}, \text{(we used (1) for the inequality)}
\]

The choice for a transcript \( t \) will be made using the following evaluation:

**DEFINITION 5.8:** We say a transcript \( t \) is **reliable for an input** \( x \) if it satisfies the conditions:

\[
i) \text{fail}_r(x, t) < \frac{1}{2^n} \quad \text{ii) break}_r(x, t) < \frac{1}{10};
\]

The 2 values \( \frac{1}{2^n} \) and \( \frac{1}{10} \) just indicate that the 2 probabilities representing failures of the GES scheme, are close to 0 [21, 34].

We can rewrite these conditions using inequality (2) and thus, we obtain that \( t \) is **reliable** when:

\[
\left\{ \begin{aligned}
\text{fail}(x, t) &< \frac{1}{2^n} \\
1 - \text{break}_r(x, t) &< \frac{1}{10}
\end{aligned} \right. \quad \Rightarrow \quad \left\{ \begin{aligned}
\text{fail}(x, t) &< \frac{9}{10} \cdot \frac{1}{2^n} \\
\text{break}_r(x, t) &< \frac{1}{10}
\end{aligned} \right.
\]

The reason why we consider these **reduced versions** of GES (where we restrict the number of random bits available for \( k() \)) is because they allow us to achieve new GES schemes with a lower failure rate, as we shall later see. But before that, we need to make some notations for the **reduced** \( G_r \), exactly as we had for the general GES: \( k^r_{x,t} = \{ k \mid t \text { is suitable for } x \text { and } k \} \), \( k^r_{x,t} \) - the **good** keys for an input \( x \) and transcript \( t \) and \( k^{w}_{x,t} \) - the **wrong** keys (which do not lead to a correct answer).

Now, we can use the properties of **reduced GES** to formulate the following relations:

We remind that \( k_r() \) generates equally probable keys all of length \( r(|x|) \).

- \( P(T_n = t \mid X = x) \) can be computed as the number of keys \( k \) such that from \( (k, x) \) we obtain the transcript \( t \), divided by the total number of keys \( k \) generated by \( k_r() \):

\[
P(T_n = t \mid X = x) = \frac{|k^r_{x,t}|}{2^{r(|x|)}}
\]

- \( 1 - \text{break}_r(x, t) = P(k() \text { uses less than } r(n) \text { bits } \mid X = x \text { and } T_n = t) \) which can be computed as the number of keys \( k \) of size less than \( r(n) \) such that \( (k, x) \) can generate \( t \), divided by the total number of keys which along with \( x \) can produce \( t \) \( \Rightarrow \) \( 1 - \text{break}_r(x, t) = \frac{|k^r_{x,t}|}{|k^r_{x,t}|} \)

- \( \text{fail}_r(x, t) \) can be computed as the ratio between the number of **wrong** keys and the total number of keys which along with \( x \) can generate \( t \) \( \Rightarrow \) \( \text{fail}_r(x, t) = \frac{|k^{w}_{x,t}|}{|k^r_{x,t}|} \)

Using these definitions and the fact that \( G_r \) hides everything but the input length, we obtain that for any 2 inputs \( x \) and \( x' \) such that \( |x| = |x'| \), we have the following property:

\[
P(T_n = t \mid X = x) = \frac{|k^r_{x,t}|}{|k^r_{x,t}|} = \frac{1 - \text{break}_r(x, t)}{1 - \text{break}_r(x, t)}
\]
What we want is to ensure that we can reduce the probability that a GES gives an incorrect answer for \( f(x) \), to a minimum, as close to 0 as possible, irrespective of the transcript between \( A \) and \( B \).

We can show the following important result [16]:

**THEOREM 10.1**: From any GES \( G \) we can construct a reduced GES \( G_r \), such that \( G_r \) satisfies the following 2 conditions:

1) \( E(fail(x,T_n)) < \frac{1}{2^{d}n} \) and 2) \( Break_r(x) < \frac{1}{n^d} \quad \forall d \text{ constant} \)

**Proof.** Given the GES definition, we know that for any GES, there exists a constant \( c \) such that \( E(fail(x,T_n)) < \frac{1}{2^{c}n} \). Thus, we only need to show that the 2 properties hold for any constant \( d > c \).

Furthermore, because \( k(\cdot) \) must run in expected polynomial time [22], there exists a polynomial \( q \) such that for any input \( x \) the expected number of bits required by \( k(\cdot) \) is less than the value of \( q(|x|) \).

Then, we consider the reduced GES \( G_z \), where the polynomial \( z \) is defined as \( z(n) = n^d \cdot q(n) \).

If we denote by \( Z \) the random variable [28] indicating the number of bits required by the function \( k(\cdot) \), we obtain that the quantity \( Break_z(x) \) associated with the scheme \( G_z \) is equal to the probability that \( k(\cdot) \) requires more than \( z(n) \) bits, which is \( P(Z > z(n)) \).

We have the following probability theory known result [34]: for any random positive variable \( X \) and for any positive real number \( t \): \( P(X > t) \leq \frac{E(X)}{t} \).

Writing this relation for our case, we have that: \( Break_z(x) = P(Z > z(n)) \leq \frac{E(Z(x))}{z(n)} \). But, by the definition of \( q \), \( E(Z(x)) = q(n) \), so we reach the inequality: \( Break_z(x) \leq \frac{q(n)}{q(n) \cdot n^d} = \frac{1}{n^d} \).

The probability that \( G_z \) outputs a wrong outcome is by definition \( \frac{1}{2} - \frac{1}{n^c} \) [16]. Therefore, we obtain \( P(G_z \text{ breaks or wrong output}) = \frac{1}{2} - \frac{1}{n^c} + \frac{1}{n^d} \) and because \( d > c \):

\[ P(G_z \text{ breaks or wrong outputs}) < \frac{1}{2}. \]

Hence, if we simulate the reduced GES \( G_z \) \( m = d \cdot n \) times and choose the answer which appears with the highest frequency in these runs, then this probability becomes less than \( \frac{1}{2} \).

Using \( G_z \), we construct a new reduced GES \( G_r \) which respects the 2 conditions of **THEOREM 10.1**.

We set \( r \) such that \( r(n) = z(n) \cdot m \forall n \).

The key-generation primitive \( k_r(\cdot) \) runs the key-generation procedure of \( G_z \) \( m \) times, thus resulting a set of \( m \) keys: \( k_1, \ldots, k_m \). For each of these keys we essentially have a different run of a GES protocol, therefore, we obtain \( m \) different transcripts at the end of which by applying the decryption algorithm of \( A \), will result \( m \) final outcomes (as candidates for \( f(x) \)). Then, we can choose as our solution for \( f(x) \) the outcome with the highest frequency amongst these \( m \) results [30].

This new defined scheme \( G_r \) respects the settings of a GES, so now we need to check if it really satisfies the 2 conditions imposed by **THEOREM 10.1**. We have already shown that if we run the \( G_z \) scheme \( m = d \cdot n \) times we obtain that \( P(fail(x,T_n)) < \frac{1}{2^{d}n} \), so condition 1 is achieved.
Additionally, the break probability of $G_r$ is equal to the probability that all $m$ runs of $G_o$ break and we saw that this probability is less than $\frac{1}{n^d}$. This concludes that condition 2) is also true, so we have constructed a GES $G_r$ originating from $G$, which satisfies both 1) and 2).

As $G$ was any arbitrary GES, this means that for every GES we can use this algorithm to obtain a new GES with the properties that: a) $E(fail(x, T_n)) < \frac{1}{2d^n}$ and b) $Break_r(x) < \frac{1}{n^d}$ for any constant $d$.

Therefore, we can assume that from now one we will work with a GES $G$ with these 2 properties.

Now from this type of GES $G$ we show the following theorem [16]:

**THEOREM 10.2**: There exists a fixed value $e$ and a number $N$, such that $\forall n > N$ we can find a set $S_n$ consisting of $n^e$ transcripts, where more than a half of the transcripts are reliable for any input $x$ of length $n$.

**Proof.** We can intuit that from this set $S_n$ we would derive our advice, as most of the transcripts in $S_n$ are reliable for all inputs. We remind that a transcript $t$ is reliable (DEFINITION 5.7) when $fail(x, t) < \frac{9}{10 \cdot 2^n}$ and $break(x, t) < \frac{1}{10}$.

Let’s first consider any possible transcript we could have for $G$. We randomly choose any such transcript $t$ and analyse the probability that $t$ is not reliable.

From property b) of $G$ we know that $E(Break(x)) < \frac{1}{n^d}$ for any $d$. We can choose $d = 2$, so we obtain: $E(Break(x)) < \frac{1}{1000}$.

Using the inequality (5) we have that $P\left(break(x, t) > \frac{1}{10}\right) \leq \frac{E(Break_r(x))}{\frac{1}{10}} \leq \frac{1}{100}$.

Similarly, from property a) of $G$, we get that $E(fail(x, T_n)) < \frac{1}{2d^n}$ for any $d$. When $d = 2$, we obtain: $E(fail(x, T_n)) < \frac{1}{2^{2n}}$.

Again, by applying inequality (5) we reach that $P\left(fail(x, t) > \frac{9}{10 \cdot 2^n}\right) \leq \frac{E(fail(x, t))}{\frac{9}{10 \cdot 2^n}} \leq \frac{10}{9 \cdot 2^n} \leq \frac{1}{100}$.

This means that the probability that $t$ is not reliable for $x$ is less than $\frac{1}{100} + \frac{1}{100}$, so the actual probability that $t$ is reliable for any $x$, $P(t$ is reliable $) > 1 - \frac{2}{100} = \frac{98}{100}$.

Now, if we have a set of $k$ independent transcripts $t_i$, we would like to see what is the probability $Pr$ that less than half of them are reliable for any input $x$. (by looking at what THEOREM 10.2 requests, we want this probability to be as close to 0 as possible).

We can view $Pr$ as the discrete probability of the number of acceptances in a row of $k$ independent yes-no trials (yes = $t_i$ is reliable, no = $t_i$ is unreliable), each being accepted a probability $p$ (in our case $p = \frac{98}{100}$). In this way, we modelled the probability $Pr$ as a binomial distribution [34], so we have that:

$Pr = P\left(number \ of \ reliable \ transcripts \ for \ x < \frac{k}{2} = F\left(\frac{k}{2}, k, p\right)\right)$, where $F$ is the cumulative distribution function for the binomial distribution [28].
Using Chernoff’s inequality [21, 34] stating that: \( F(m, n, p) \leq \exp\left(-\frac{1}{2p} \cdot (np-m)^2 \right) \), we obtain:

\[
P\left(\text{number of reliable transcripts for } x < \frac{k}{2}\right) < \exp\left(-\frac{1}{2} \cdot \frac{\left(98k \cdot 100 - \frac{1}{2}\right)}{k}\right) = \frac{1}{2^{\alpha(k)}}
\]

Therefore, for a set \( S \) of size \( k = n^e \) transcripts, \( e \geq 1 \) constant, this probability is less than \( \frac{1}{2^{\alpha(n^e)}} \). But as there are \( 2^n \) possible inputs of length \( n \), when we cumulate this probability for every input we reach that: \( P\left(\text{number of reliable transcripts for all } x < \frac{|S|}{2}\right) < 2^n \cdot \frac{1}{2^n} = 1 \).

Then, since the probability is strictly less than 1, this means that there must exist a set \( S_n \) satisfying the conditions mentioned in THEOREM 10.2. 

**OBS 6:** We make the next important observation: for any reliable transcript \( t \) from \( S_n \) we have an associated set of keys \( k \) which could produce \( t \), namely \( k(x, t) \). But, in order to reach the correct result \( f(x) \), we need to make a distinction between the subsets of good keys (the ones that lead to \( f(x) \)) and the set of wrong keys (the ones which lead to an incorrect answer). Then, we could use the good keys corresponding to the reliable transcript \( t \), as part of the external advice to solve \( f(x) \) in NP.

This separation between good and wrong keys is achieved through the use of Universal Hashing [35].

**DEFINITION 5.9:** Universal hashing can be described as follows: we have \( H \), a set of independent hash functions that map elements from any set \( S \) to the set \( T = \{0, ..., m - 1\} \). \( H \) has the important property that by randomly choosing any function \( h \) from \( H \), we have a small number of collisions:

\[
P(h(a) = h(b)) < \frac{1}{m} \quad \forall a, b \in S.
\]

In our case we use the universal hashing \( H \) to map keys of \( G \) to the set \( T = \{0, ..., m - 1\} \).

Consequently, using the collision property of \( H \) [35], we have that for a subset of \( p \) keys, \( \{k_1, ..., k_p\} \) and \( h \in H \), the probability that all \( p \) keys are mapped to exactly the same value is:

\[
P(h(k_1) = ... = h(k_p)) = \frac{1}{m} \cdot \frac{1}{m} ... \frac{1}{m} = \frac{1}{m^{p-1}}.
\]

**DEFINITION 5.10:** We name a set of \( p \) keys \( L = \{k_1, ..., k_p\} \) for which we have \( h(k_1) = ... = h(k_p) \), a \((h, p)\)-collision set.

The separation mentioned before, between good and wrong keys is based on the following result [16]:

**THEOREM 10.3** For any transcript \( t \) there exist 2 numbers \( m \) and \( p \) and a function \( h \in H \) such that:

(A) The set of good keys \( k g_{x,t} \) includes a \((h, p)\)-collision set, for any \( x \) such that \( t \) is reliable for \( x \);
(B) The set of wrong keys \( k w_{x,t} \) cannot include any \((h, p)\)-collision sets, for any \( x \) such that \( t \) is reliable for \( x \);

**Proof.** We notice that if the number of good keys, \(|k g_{x,t}|\) is larger than \( p \cdot m \), then any hash function \( h \) must map at least \( p \cdot m \) keys to the \( m \) elements of \( T \) [34]. In this case, there needs to exist at least \( p \) keys such that \( h \) maps them to the same element of \( T \), which would result in a \((h, p)\)-collision set.

To satisfy the (B) condition we must analyse the probability that for any randomly chosen \( h \in H \), \( h \) can produce a \((h, p)\) – collision set. After that, we determine values for \( m \) and \( p \) such that this probability is strictly less than 1.
Consider the class of all sets of wrong keys $k_{W,i,t}$ corresponding to a $x_i$ such that $t$ is reliable for $x_i$. We set $U$ to be the maximum of the cardinalities of these sets.

For a given $x_i$, we can choose a subset of $p$ keys from a maximum number of $U$ keys in $\binom{U}{p}$ ways.

The probability that any of these subsets could form a $(h, p)$-collision set becomes $\binom{U}{p} \cdot \frac{1}{m^{p-1}}$ (by the definition of $H$).

As the total number of inputs $x_i$, such that $t$ is reliable for them, is at most $2^n$, we obtain that the probability of having a $(h, p)$-collision set containing only wrong keys from $k_{W,i,t}$ is less than $\binom{U}{p} \cdot \frac{1}{m^{p-1}} \cdot 2^n$.

This means that we need to find values for $p$ and $m$ such that this quantity is strictly less than 1.

Using Stirling’s formula [21] we have that $M = \binom{U}{p} \cdot \frac{1}{m^{p-1}} \cdot 2^n < \left(\frac{eU}{p}\right)^p \cdot \frac{1}{m^{p-1}} \cdot 2^n = m \cdot 2^n \cdot \left(\frac{eU}{pm}\right)^p$.

By setting the value of $m$ such that $eU < m < 2eU$, we obtain that $M < 2^n \cdot 2eU \cdot \frac{m^p}{p^p} = \frac{2^{n+1}eU}{p^p}$.

Because every key $k$ must be of polynomial size, there exists a constant $q$ such that $k < n^q$. Then, since $U$ is a subset of all the possible keys, we have $U < 2^{n^q}$.

Therefore, $M < \frac{e \cdot 2^{n+1+n^q}}{p^p}$. Now, we just need to set the value of $p$ such that $M$ is always strictly less than 1:

$$\frac{e \cdot 2^{n+1+n^q}}{p^p} < 1 \Leftrightarrow 2^n + n + e\log(e) - \log(p) < 1.$$ So, by setting $p = n^{q+1}$ we have satisfied condition (B).

We observed before that in order to meet condition (A) we must make sure that $|k_{W,i,t}| > p \cdot m$.

To verify this we use the relation (4) which specifies that for any 2 inputs $x, x'$ we have:

$$\frac{|k_{W,i,t}|}{|k_{W,i',t}|} = \frac{1 - \text{break}(x,t)}{1 - \text{break}(x',t)}.$$

We choose $x$ and $x'$ in the following way:

- $x$ is the input $x_i$ such that $t$ is reliable for $x_i$ and the corresponding set of keys $k_{x_i,t}$, has the property that $|k_{x_i,t}| = \min_{x_j} |k_{x_j,t}|$
- $x'$ is the input $x_i$ such that $t$ is reliable for $x_i$ and the corresponding set of keys $k_{x_i,t}$, has the property that $|k_{x_i,t}| = \max_{x_j} |k_{x_j,t}|$

Then, relation (4) becomes:

$$\frac{\min_{x_j} |k_{x_j,t}|}{\max_{x_j} |k_{x_j,t}|} = \frac{1 - \text{break}(x,t)}{1 - \text{break}(x',t)}$$

which is equivalent to:

$$\min_{x_j} |k_{x_j,t}| \cdot \frac{\max_{x_j} |k_{x_j,t}| \cdot (1 - \text{break}(x,t))}{1 - \text{break}(x',t)} \geq \max_{x_j} |k_{x_j,t}| \cdot (1 - \text{break}(x,t))$$

Additionally, we know that for any transcript $t$ of $G$ we have that $\text{break}(x,t) < \frac{1}{10}$, thus we obtain:

$$\min_{x_j} |k_{x_j,t}| > \frac{9 \max_{x_j} |k_{x_j,t}|}{10}.$$ By making the notation $V = \max_{x_j} |k_{x_j,t}|$, the inequality becomes: $\min_{x_j} |k_{x_j,t}| > \frac{9V}{10}$. 

35
Now, the quantity we are interested in, \(|k g_{x,t}|\) is the difference between the total number of keys and the number of wrong keys:

\[|k g_{x,t}| = |k x_{x,t}| - |kw_{x,t}| \geq \min_{x_j} |k x_{x,t}| - \max_{x_j} |kw_{x,t}| = \min_{x_j} |k x_{x,t}| - U > \frac{9\sqrt{v}}{10} - U\]

Using that \(\text{fail}(x,t) < \frac{1}{2^n}\) and relation (3) we get that: \(\frac{U}{v} < |kw_{x,t}| \leq \frac{1}{2^n}\).

Therefore, \(|k g_{x,t}| > \frac{9\cdot\sqrt{v} - U}{10} = U\left(\frac{9\cdot\sqrt{v}}{10} - 1\right)\). From \(eU < m < 2eU\), we finally reach:

\[|k g_{x,t}| > \frac{m}{2e} \cdot \left(\frac{9\cdot\sqrt{v}}{10} - 1\right) > pm,\] which is exactly what we wanted to show.

We now discuss the implications of \textsc{THEOREM 10.3}.

\textbf{OBS 7}: Consider any \textit{reliable} transcript \(t\) selected from \(S_n\) and \(C\) a \((h,p)\)-collision set as constructed above. \textsc{THEOREM 10.3} tells us there are 2 possibilities:

\begin{itemize}
  \item[i)] either the set of \textit{good} keys corresponding to \(t\) contains the subset \(C\) of \(p\) keys all mapped by \(h\) to the same value or
  \item[ii)] \(C\) contains both \textit{good} keys and \textit{wrong} keys (because the second case of \textsc{THEOREM 10.3} tells us there can be no \((h,p)\)-collision set containing only \textit{wrong} keys).
\end{itemize}

Therefore, for any \((h,p)\)-collision set \(C\) we can verify if \(C \subseteq k(x,t)\). If the \textit{outcomes} obtained for \(f(x)\) from all keys in \(C\) are \textit{equal} (meaning that all keys in \(C\) are \textit{good}), then we can conclude that we are in case i) and we know for sure that this \textit{outcome is the real} \(f(x)\).

If we were in case ii), because \(C\) had both \textit{good} and \textit{wrong} keys, we would have at least 2 different outcomes for \(f(x)\) resulting from 1 \textit{good} key and 1 \textit{wrong} key.

\subsection*{Algorithm to solve \(f\)}

Finally, we are able to define the \textsc{NP/poly} algorithm which computes \(f(x)\).

As suggested before, the advice received for all inputs of length \(n\) consists of the set of transcripts \(S_n = \{t_1, ..., t_w\}\) and for each transcript \(t_i\) the advice also contains the associated universal hash function \((h_i, p_i, m_i)\).

The \textsc{NP} algorithm can be described as follows:

\begin{itemize}
  \item We receive the advice \(S_n\) containing a majority of \textit{reliable} transcripts.
  \item The nondeterministic step tries every subset of \(S_n\) of size larger than \(\frac{|S_n|}{2}\) until we find the one consisting of the \textit{reliable} transcripts of \(S_n\).
  \item Now, for each transcript \(t_i\) from this subset we must find a \((h_i, p_i)\)-collision set \(C = \{k_{i_1}, k_{i_2}, ..., k_{i_{p_i}}\}\) containing only \textit{good} keys. We use another \textbf{nondeterministic choice} to identify \(C\). To check that \(C\) is a \((h_i, p_i)\)-collision set, we look at the mappings \(h_i(k_{i_j})\) and we test if they are all equal. Then, as described in \textbf{OBS 7}, we know that \(C\) only consists of \textit{good} keys if the outcomes obtained for \(f(x)\) are all equal to the same value, call it \(y_i\).
  \item Finally, we verify that for all transcripts \(t_i\) we have the \textbf{same outcome} \(y_i\). We conclude that this value is exactly the \textbf{desired} \(f(x)\).
\end{itemize}

This algorithm ends the proof for the \textbf{MAIN THEOREM 1} \([16]\) by showing that \(f\) can be computed by an \textsc{NP/poly} algorithm.
We have seen which are the implications for solving any problem \( f \) using the GES framework. Our main target is to decide whether a function \( f \in BQP \) could be computed in this scenario. But before that, we investigate a similar protocol called Universal Blind Quantum Computing, which has similar aim: to securely delegate a quantum computation. One of the main reasons why we consider this protocol is because Universal Blind Quantum Computing can solve any BQP problem [15].

Consequently, we will investigate and in fact develop an adaptation of the GES scheme in order to match the Universal Blind Quantum Computation definition.

5. Universal Blind Quantum Computing

Universal Blind Quantum Computing (UBQC) is a quantum protocol proposed by Broadbent, Fitzsimons and Kashefi [26], in which a client with limited computational resources asks a server owning a quantum computer to solve a problem for him. The client gets the result of his computation in such a way that the server cannot infer anything about the input or even about the computation required by the client. In this scheme, the client must be able to prepare single qubits and then send them to the server, but does not need any other quantum resources.

UBQC is an interactive protocol where the client and server exchange messages [11, 12]. In order to obtain the desired computation outcome, the client must indicate to the server what qubit measurements to perform depending on past measurements results. Additionally, the scheme also allows the client to address a problem which has either classical or quantum inputs/outputs [15].

UBQC protocol is constructed using the Measurement Based Quantum Computing model [29]. It creates a generic pattern for all quantum computations and sets apart the quantum and classical components of a given problem. As a result, the client does not need to make any quantum computation or to own quantum memory.

Furthermore, we can also identify a malicious server which might try to deceive the client. Keeping the input and the computation private from the server is not based on the difficulty of solving hard problems (as it is the case for current classical cryptographic systems [9]) and its security can be maintained irrespective of what a malicious server might try to do. If the client would require the solution for an \( NP \) problem, then he could check if the result received from the server is correct (solutions to \( NP \) problems can be verified in polynomial time [17]). In this way, he would find out if the server should be trusted or not. However, for harder problems and in the cases where the server interferes with the computation [12], the client may be deceived.

This scheme is universal, meaning that it can work for any quantum computation [6]. For instance, suppose the client would like to use the computational power of the server for the problem of factorization. Then, using the Shor’s algorithm [7] in the form of a blind computation, will allow the client to obtain the factorization, without the server knowing the input or even the fact that he is factorizing a number. In fact, this scheme can be used for any BQP – complete problem, such as \( k \)-Forrelation [25](which will be discussed later).
UBQC can be understood as an interactive Measurement Based Quantum Computing [26], where both the client and the server contribute, in order to perform the computation:

1) The client needs to build qubits
2) The server generates the entangled resource and completes the measurements
3) The client is instructing the server upon the next measurements and is also adding some randomness to keep the computation private from the server.

Now, suppose in the MBQC model the client would use the highly entangled graph state [1, 29]. As we remember, the graph state consisted of multiple qubits from which we derived our computation by making measurements in either the \(|0\rangle, |1\rangle\) or in the \(|\pm \theta\rangle\) basis [30]. The problem with this approach is that we might need different graphs for different problems, hence the client would leak some information about the desired computation.

Instead of working on a graph state, we use as starting point a brickwork state (Fig. 3) [26]. This state has the advantage that by performing only measurements in the \(|\pm \theta\rangle\) basis we can obtain any quantum operator. Even more, this state has the same pattern for all quantum computations and only depends on the size of input and size of computation. In the MBQC scenario, the measurements destroyed the initial graph state (removing certain qubits from the graph), so a malicious server can deduce the pattern of the graph. On the other hand, in the situation of using the brickwork state we only need to measure in the \(|\pm \theta\rangle\) basis, so the brickwork state is not altered, remaining generic for all computations. Thus, it does not leak any information about the problem [15, 27].

Now, we will present how does the Universal Blind Quantum Computing work and what are the properties of this protocol.

Suppose the client $A$ has a problem $\mathcal{P}$, which he wants to solve using UBQC. Then, $A$ needs to have a description of the circuit using the brickwork state $G$ (Figure 3). Client $A$ randomly constructs qubits...
in the states \[ \frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi}|1\rangle) \mid \phi \in \left\{ 0, \frac{\pi}{4}, \ldots, \frac{7\pi}{4} \right\} \] and delivers them to the server B. Once B
receives the qubits, he must entangle them in accordance with the state \( G \).

The brickwork state can be viewed as a matrix of qubits. For each line and for each qubit of the line, 
A will tell B which is the angle of the measurement for that particular qubit [27]. As a result, B
completes the measurement and announces A about the result. Then, A will do the same thing for
the next qubits, but their angles determined by A, are based on the previous results received from B.
A obtains the desired computation result once all the qubits in the brickwork state are measured [15].

Consider that for the problem \( \mathcal{P} \) we have the brickwork state \( G \) having \( l \) lines and \( c \) columns. Then, 
each qubit \( |\phi_{l,j}\rangle \) of this state is initially in the \( |+\rangle \) state and it has associated a measurement angle \( \rho_{l,j} \). As in the MBQC model, these angles are affected by previous measurements: any \( \{|+\theta\rangle, |-\theta\rangle\} \)
measurement implies possible \( X \) corrections changing an initial angle \( \rho \) to \( -\rho \) and \( Z \) corrections
changing an initial angle \( \rho \) to \( \rho + \pi \) [15]. Therefore, we have that the actual angles for every qubit are:

\[ \rho'_{l,j} = (-1)^s \rho_{l,j} + t\pi, \text{ where } s \text{ is the number of previous } X \text{ corrections and } t \text{ is the number of previous } Z \text{ corrections [29].} \]

We can view the problem \( \mathcal{P} \) as a quantum operator \( U \) and so, the client \( A \) wants to obtain the result
\( U|0\rangle \). The UBQC scheme for computing \( U \) is the following [26]:

- **Stage 1: Qubits Setting**

  For every column \( c \) of \( G \) and for every line \( l \) of \( G \):
  
  Client \( A \) sets the qubit \( |\phi_{c,r}\rangle \) randomly chosen from the set

  \[ S = \left\{ \frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta_{l,j}}|1\rangle) \mid \theta_{l,j} \in \left\{ 0, \frac{\pi}{4}, \ldots, \frac{7\pi}{4} \right\} \right\} \]

  and sends \( |\phi_{c,r}\rangle \) to the server \( B \);

  Using the qubits obtained from the client, the server builds the entangled state. This is achieved
by applying the operator Controlled-Z between pairs of qubits, thus defining the structure of \( G \).

- **Stage 2: Qubits measurement and communication client-server**

  For every column \( c \) and line \( l \) of \( G \):

  The client computes the standard angle of measurement \( \rho'_{l,j} \) in order to obtain \( U \)

  \( A \) also adds an element of randomness to hide the computation from \( B \), \( rand \in \{0,1\} \) and sets
the angle of measurement as: \( \delta_{l,j} = \rho'_{l,j} + \theta_{l,j} + rand \cdot \pi. \)

  This new angle is received by the server \( B \). Consequently, \( B \) measures the qubit in the basis

  \[ \frac{1}{\sqrt{2}}(|0\rangle + e^{i\delta_{l,j}}|1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - e^{i\delta_{l,j}}|1\rangle) \]

  and obtains the result \( r \in \{0,1\} \) which is sent to the client.

  The client needs to modify the result \( r \) to compensate for the randomness he added, so the result
  of the measurement becomes \( r' = |rand - r| \)
The **Universal Blind Quantum Computing** protocol has 3 important properties: **universality**, **correctness** and **blindness** [6, 9], which we will next define.

**DEFINITION 6.1:** **Universality** refers to the fact that the scheme allows the computation of any quantum operator [6]. This is achieved due to the fact that the MBQC model is universal and in fact we can obtain this property by only performing measurements with the angles: $\{-\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}\}$. □

**DEFINITION 6.2:** Showing that the result returned by the protocol is indeed $U|0\rangle$ means the protocol is correct [15].

The stage 1 of the protocol involves the rotations of the qubits around the $Z$-axis with the angle $\theta$ and the application of the Controlled-$Z$ gate on pairs of qubits. These operations do not affect the brickwork state (as opposed to the graph state case), but only modify the phase of qubits as if the Controlled-$Z$ gate was first applied and then the qubits were rotated with the angle $\theta$.

Furthermore, $A$ changing the angle of measurement from $\rho'$ to $\rho' + \theta$ is equivalent to adjusting the state on which we applied the measurement from $|\phi\rangle$ to $Z(\theta)|\phi\rangle$. Therefore, as $A$’s choice for the new measurement angle is defined as: $\delta_{i,j} = \rho'_{i,j} + \theta_{i,j} + rand \cdot \pi$, then if $rand = 0$ the server’s outcome $r$ is exactly the result $A$ should have and if $rand = 1$, then $A$ just needs to change the result to $1 - r$. □

**DEFINITION 6.3:** The **blindness** property refers to the fact that $B$ cannot infer anything about $A$’s input or the actual computation (except their size), no matter what $B$’s actions might be [26].

We can view this task of blind computation in the following perspective: the client $A$ needs to solve the problem $\mathcal{P}$ by exchanging messages with the quantum server $B$. Then, for a given input $x$ and the description of $\mathcal{P}$ as an operator $U$, $A$’s input for the protocol becomes $I = (U, x)$.

**DEFINITION 6.4:** We say a delegated quantum computation acting on input $I$ is **blind hiding everything except $H(I)$** [27] if the knowledge inferred by a malicious server $B$ has no connection with the input $I$ and the quantum state of $B$ is also uncorrelated with $I$.

**THEOREM 11.** We are now able to show that the UBQC scheme is a delegated quantum computation hiding everything but the size of the brickwork state [15] (the number of lines $l$ and the number of columns $c$).

**Proof.** As we have already shown, the generic brickwork state allows the computation of any operator $U$, without being modified in any way by the measurements. Therefore, we do not reveal any information about $U$, except the size of the problem, $l$ and $c$.

The client $A$ has the initial measurement angles $\rho_{i,j}$ for every qubit, needed to obtain $U$. Then, these angles are altered based on the outcome of the previous measurements executed by $B$, to form the new angles: $\rho'_{i,j} = (-1)^{\delta_{i,j}}\rho_{i,j} + \pi \cdot \theta$. $A$ does not send to $B$ exactly these angles, but instead, the classical knowledge $B$ obtains is represented by the measurement angles: $\delta_{i,j} = \rho'_{i,j} + \theta_{i,j} + rand \cdot \pi$. 

40
Therefore, what we need to prove is that $\delta_{i,j}$ (the classical knowledge of $B$) is not connected with $\rho_{i,j}$ (the input of the problem).

We know the fact that $\theta_{i,j}$ is randomly drawn from the set $\left\{0, \frac{\pi}{4}, \ldots, \frac{7\pi}{4}\right\}$. Thus, because $\delta_{i,j} = \rho_{i,j}' + \theta_{i,j} + \text{rand} \cdot \pi$, we have that $\delta_{i,j}$ cannot be based on $\rho_{i,j}$ [29].

At this point, we analyse the quantum state of $B$ resulted after $A$’s qubits preparation, call it $S$.

For each qubit of $S$, $|\phi_{i,j}\rangle$ we need to consider 2 cases:

- If $\text{rand} = 0$, then the angle received by $B$ is $\delta_{i,j} = \rho_{i,j}' + \theta_{i,j}$ and thus, the qubit state is:
  
  $|\phi_{i,j}\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\theta_{i,j}} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + e^{i(\delta_{i,j}-\rho_{i,j})} |1\rangle)$

- If $\text{rand} = 1$ then the angle received by $B$ is $\delta_{i,j} = \rho_{i,j}' + \theta_{i,j} + \pi$ and thus, the qubit state is:
  
  $|\phi_{i,j}\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i(\theta_{i,j}+\pi)} |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle - e^{i(\delta_{i,j}-\rho_{i,j})} |1\rangle)$

We observe that if we fix the value of $\delta_{i,j}$, then $\theta_{i,j}$ will depend on $\rho_{i,j}'$ ($\theta_{i,j} = \delta_{i,j} - \theta_{i,j} + \text{rand} \cdot \pi$). However, $\text{rand}$ is chosen independently at random from $\{0, 1\}$, therefore $S$ contains qubits from the brickwork state, which are not based on the input $\rho_{i,j}$.

We can now illustrate the usefulness of this blindness property of UBQC by presenting 2 attack examples [26, 27].

**Example 1:** Assume $B$ owns some additional prior information about the input $I$. But using the blindness definition, it results that this prior information is not connected with $I$ and therefore, the only knowledge inferred by $B$ is the leaked information, the size of the computation.

**Example 2:** Assume $B$ aims to find some information about the output of the computation. However, because the output would be based on the input $I$ and we know that $B$ cannot find out anything about the input, then this situation is also impossible.
6. Classical Universal Blind Quantum Computing

In the Universal Blind Quantum Computing protocol, the client does not need to possess quantum memory or the ability to perform quantum computations, however, he must be able to prepare qubits in different states, so he is not an entirely classical client [26, 29].

What we want to investigate is the possibility of the existence of a “Classical Universal Blind Quantum Computing” (CUBQC) scheme, where the client is purely classical and where the interaction between the client and the quantum computer must also be classical.

A similar scheme for this type of blind delegated computation involving one entirely classical client was proposed by Broadbent et al [15]. Unfortunately, this protocol requires 2 entangled quantum servers: the first one completes the quantum work done by the client in the UBQC scenario (prepares randomly selected qubits) while the second one performs the actual computation.

This classical UBQC protocol which we propose must obey the UBQC scheme definition according to whom the input, output and the computation itself must be private to the quantum server. Moreover, it should respect the 3 main properties discussed before: universality, correctness and blindness [27].

We relate the existence of CUBQC with the Generalised Encryption Scheme. To be more specific, we will prove that CUBQC can be represented as an instance of the GES framework and thus, this would imply that any major result known for GES is also applicable to CUBQC.

To begin with, by comparing CUBQC with GES, we notice that the major difference between the two interactive delegated computation schemes is the following:

In GES, the server already knows the problem needed by the client and we use the encryption procedure to hide only the input and the output of a problem, whereas in UBQC the actual computation is also kept secret from the server.

Therefore, we need to make the modifications such that in the GES scenario we also encode the actual computation in the input string given to the encryption algorithm, with the resulting ciphertext being sent afterwards to the server.

At this moment is worth mentioning a brief review of the main GES principles [16]:

\[ \text{Client } A \xrightarrow{y=E(x)} \text{Server } B \]

\[ \text{Client } A \xleftarrow{f(y)} \text{Server } B \]

The client \( A \) needs to determine the value of a function \( f \) applied on an input \( x \). He encrypts this input, \( y = E(x) \), and sends the encryption \( y \) to the server. The server then applies the function \( f \) on \( y \) and sends the result back to \( A \). Finally, \( A \) only needs to apply the decryption function to obtain \( f(x) \). Therefore, the encryption and decryption primitives must satisfy the important property that:

\[ D \left( f(E(x)) \right) = f(x). \]

Furthermore, because the client \( A \) has only limited computational resources the encryption and decryption functions must run in polynomial time.
Before proceeding to the main proof, we need to outline some essential notions which will be used in our arguments.

DEFINITION 7. A Quantum Turing Machine (QTM) is the correspondent in the quantum perspective of the Probabilistic TM [6]. The classical probabilistic machine evolves according to a probability distribution describing the likelihood of every future state of the machine. In the quantum machine case, we are dealing with a quantum superposition of possible states where every coefficient of a state represents its amplitude [10].

The internal states of a QTM M are represented by states in the Hilbert Space [1], while the transition function is a unitary operator. M can be described by the tuple \((\Sigma, Q, \delta, q_0, q_f)\), where \(\Sigma\) is the set of input symbols, \(Q\) is \(M\)’s set of states from the Hilbert Space, \(\delta\) is the transition function of the machine, \(q_0\) is the initial state and \(q_f\) is the accepting state.

\(M\) halts on an input \(x\) if the machine state gets in a superposition of possible configurations, all in the accepting state \(q_f\). We say that \(M\) is a polynomial-time Quantum TM if it halts on every possible input \(x\) and the running time of the machine is polynomial in \(|x|\) [10, 17].

Using these descriptions, we are now ready to define a Universal Quantum Turing Machine (the quantum version of the classical Universal TM).

A Universal Quantum Turing Machine would allow us to simulate any QTM \(M\), by describing the specification of \(M\) and sending it as an input to the universal machine.

Bernstein and Vazirani proved the existence of this generic machine [10], we will call it \(M_U\). Specifically, \(M_U\) receives as input: the representation of a QTM \(M\), an input \(x\) for \(M\) and a precision \(E\) and returns a quantum superposition \(\psi\), such that the distance between \(\psi\) and the superposition of \(M\) applied on \(x\) is less than \(E\).

The way \(M_U\) functions is very straightforward: it is based on a deterministic Turing Machine extended with only one quantum procedure: a quantum analogue of a coin flip which rotates individual qubits [1]. It was proved that in order to simulate any quantum TM it suffices to apply this operation using the amplitudes \(\frac{3}{5}\) and \(\frac{4}{5}\) [30]. Additionally, the running time of \(M_U\) is polynomial in the length of the input.

What is left, is to define the representation of any QTM as a string which would then be given as part of the input to \(M_U\). For a given QTM \(M\) we can encode it in a string \(S\) which contains the following: the finite alphabet \(\Sigma\), the number of states from \(Q\) and the description of the transition function \(\delta\).

Every input of the \(\delta\) function consists of a tuple \((q_1, s_1, s_2, q_2, d)\), where \(q_1\) represents the current state of \(M\), \(s_1\) is the symbol on the infinite tape, indicated by the tape’s head, \(s_2\) is the symbol \(M\) will write on the tape, \(d\) expresses whether the head of the tape is moving left, right or remains at the same position and \(q_2\) is the next state of \(M\). The transition can be illustrated like this: suppose \(M\) is in state \(q_i\). Then, at next step the machine will be in the quantum superposition \(\psi = \sum_j q_j \cdot \alpha_j\), with \(q_j\) representing a possible future state and \(\alpha_j\) the amplitude associated with a transition tuple \((q_i, s_{i,1}, s_{i,2}, q_j, d)\) \((\alpha_j^2\) is the probability of this transition) [20, 30]. Therefore, the encoding string \(S\) must contain the amplitudes associated with all these possible tuples (which are in number of \(3 \cdot |Q|^2 \cdot |\Sigma|^2\)).
Now we can return to our problem and describe one of our primary contributions.

**THEOREM 12.** CUBQC can be represented as an instance of the GES protocol.

**Proof.** In the GES scheme the server holds the function $f$ and he applies it on whatever input (encrypted input) he receives from the client.

We change $f$ so that instead of denoting the computation required by the client, $f$ will represent a Universal Quantum Turing Machine $M_U$.

Suppose the client wants to compute a problem $P$ for the input $x$. We know that there must exist a QTM $M$ which is able to solve $P$. Then, we can encode the behaviour of $M$ in a string $S$, using the method described above. It is worth mentioning that the encoding can be done efficiently in polynomial time, so the client having only BQP computational power can perform this procedure.

Then, the client's input $I$ would consist of $S$ and the input $x$ for $M$, $I = (S, x)$. From this point, the client proceeds as in the GES scheme. He applies the encryption primitive $E$ to obtain $J = E(I)$ and sends $J$ to the server.

The server uses $J$ as input for the generic UQTM $M_U$ and sends back to the client the outcome returned by $M_U$. Finally, the client applies the decryption function on the outcome received from the server and obtains the desired result $f(x)$.

The encryption and decryption procedures must satisfy the property: $D(M_U(E(S, x))) = f(x)$. (*)

In this way, by using the Universal QTM we assured that the server cannot find anything about the computation needed by the client, since the problem is also sent to the server as an encrypted input.

We need to verify if the 3 properties assured in the UBQC protocol are also available here.

The correctness property, which refers to the fact that the client obtains in the end $f(x)$, is guaranteed by the GES scheme, specifically by the decryption and encryption functions property (*)&.

Universality is obtained because the UQTM $M_U$ allows the simulation of any QTM $M$, therefore the client can use this scheme for any computation [10].

Because the GES protocol has the privacy property that the server cannot infer anything about his input except its length [16], this implies that CUBQC is blind, leaking only the size of the computation and the length of the input, which is exactly the level of security guaranteed in the UBQC protocol.

Consequently, we defined the Classical UBQC scheme, represented as an instance of GES.

Therefore, any complexity theoretic result regarding classes of problems which can or cannot be solved using GES also applies to the CUBQC protocol.

Let's consider that the client $A$ wants to compute a problem $P$ on an input $x$, where $P$ is from the BQP class. Because $P \in BQP$, there exists a polynomial-time QTM $M$ which can solve $P$ [19]. Then, $A$ encodes the representation of $M$ and obtains the string $S = \rho(M)$. $A$ encrypts $S$ along with the input $x$ and sends the resulting encryption $e$ to the server $B$. $B$ runs the UQTM $M_U$ on the input $e$ and sends back to $A$ the outcome of $M_U$. In the end, $A$ decrypts this outcome and gets $P(x)$. However, we will next present strong evidence that the GES scheme cannot solve $BQP - hard$ problems. This tells us that the no-go result also stands for Classical UBQC, namely that there does not exist any CUBQC scheme which could solve for $BQP - hard$ problems.

44
Our Main Results

Next, we will proceed to the central part of the research project which depicts our most important contribution: analyse the possibility of $BQP$ — hard problems being solved using $GES$.

We base our proofs on the main result of Abadi obtained for $GES$ (described in MAIN THEOREM 1), according to which any problem solvable using $GES$ must belong to the class $NP/poly \cap coNP/poly$. Then, the statement “No $BQP$ — hard problems can be solved using $GES$” is equivalent to showing that $BQP \subseteq NP/poly \cap coNP/poly$. Our target is to give strong evidence for this major complexity-theory result.

In the following 2 sections, we explore two problems which indicate the oracle separation between the classes $BQP$ and $NP/poly \cap coNP/poly$, namely: Simon’s problem and $k$-Forrelation.

7. Simon Problem

Simon problem is a decision problem involving a “black-box” [21] function $f$, where we need to determine what kind of function is $f$ given a promise about it. More exactly, we are promised that $f$ is either 1-to-1 or 2-to-1 and we need to determine whether $f$ is of the former or latter type [23].

DEFINITION 8.1: We call a function $f: \{0,1\}^n \rightarrow \{0,1\}^n$ 1-to-1 if $f$ is bijective.

DEFINITION 8.2: We say a function $f: \{0,1\}^n \rightarrow \{0,1\}^n$ is 2-to-1 if it satisfies the following property:

$\exists s \in \{0,1\}^n - \{0\}$ such that $\forall x, x'$ we have: $f(x) = f(x') \iff x' = x \oplus s$. We call $s$ the xor mask of $f$.

$f$ is a “black-box” function refers to the fact that $f$ is given in the form of an oracle (we cannot see its explicit definition, but we can ask the oracle for values of $f$ at different points).

Classical Algorithm for Simon

In order to solve this problem deterministically, we would need $O(2^n)$ queries [23]. For instance, we could search for an element $m, 1 \leq m \leq 2^n - 1$ such that $f(0) = f(m)$. If we find such an $m$, then $f$ must be 2-to-1 (it cannot be bijective) and its xor mask is $m$. Otherwise, if no such $m$ exists we conclude that $f$ must be bijective. However, using a classical randomized algorithm we can obtain a better time-complexity [28], solving this problem with only $O(\sqrt{2^n})$ queries. We pick some input values $x_i$ at random, and for each, we consult the oracle to get $f(x_i)$. Once we find a collision pair, namely 2 input values $x_i$ and $x_j$ such that $f(x_i) = f(x_j)$, we know $f$ is 2-to-1 and $s = x_i \oplus x_j$. Using the “birthday paradox”$^4$ [34] we achieve that with only $\sqrt{2^n}$ queries the problem can be solved. Moreover, it can also be proven that since $f$ can only be accessed through the oracle that computes its values, the classical query complexity $\sqrt{2^n}$ is optimal.

$^4$ The probability that 2 persons have the same birthday. Given there are 365 days in a year and birthdays are almost uniformly distributed, what matters is the number of pairs of people: if we have $\sqrt{n}$ persons, where $n$ is the number of days in a year, there are $n$ different pairs of people who can have the same birthday and with high probability one of them will succeed.
7.1. Quantum Algorithm

In the quantum paradigm, this problem can be solved using \( n \) queries to the oracle \( O \), thus showing an exponential speedup compared to its classical probabilistic counterpart [30].

**Figure 4. Quantum Circuit for Simon's Algorithm**

We can prove our problem is in the \( BQP \) class by presenting an expected quantum polynomial-time algorithm which always gives the correct answer for Simon problem.

**MAIN THEOREM 2.1. Simon Problem \( \in BQP^O \)**

**Proof.** In order to solve this problem, we are going to use Simon’s algorithm [23] (Figure 4).

The initial state is \( |\psi_0\rangle = (|0\rangle \otimes \ldots \otimes |0\rangle) \otimes (|0\rangle \otimes \ldots \otimes |0\rangle) = |0\rangle^\otimes n \otimes |0\rangle^\otimes n \) (2 registers of \( n \) qubits)

The first thing we do is create an equal superposition over all \( 2^n \) possible inputs to \( f \) and we achieve this by applying \( n \) Hadamard operators on the first register.

After applying the \( n \) Hadamard gates on the first \( n \) qubits (first register), we get the state:

\[
|\psi_1\rangle = +^\otimes |0\rangle^\otimes n = \frac{1}{\sqrt{2^n}} (|0\rangle + |1\rangle)^n |0\rangle^\otimes n = \frac{1}{\sqrt{2^n}} \left( \sum_{x \in \{0,1\}^n} |x\rangle \right) \otimes |0\rangle^\otimes n
\]

The next step is querying \( f \).

Applying the black-box operator for \( f \) has the following effect [29]:

\[
|x\rangle \xrightarrow{U_f} |x\rangle \quad |y\rangle \xrightarrow{U_f} |y \oplus f(x)\rangle
\]

Thus, in our case we obtain:

\[
|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes 0^n \oplus f(x) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle
\]

- we have reached a superposition of all values of \( f \) in the second register.
As we can see in the circuit, in state $\ket{w_2}$ we will measure the second register which contains $f(x)$.

After the measurement, what is left in the first register, will be a superposition over all possible inputs $x$ that could have produced the $f(x)$ we observed. This superposition is $\frac{1}{\sqrt{2}} (\ket{x} + \ket{y})$ where $x$ and $y$ are 2 inputs with the property that $f(x) = f(y)$.

Next, we apply $n$ Hadamard gates on the first register (which now only consists of $\frac{1}{\sqrt{2}} (\ket{x} + \ket{y})$). Applying Hadamard on an input $\ket{x}$ consisting of $n$ qubits has the following effect [1]:

$$|x\rangle \xrightarrow{H^n} \frac{1}{\sqrt{2^n}} \sum_{w \in \{0,1\}^n} (-1)^{x \cdot w} |w\rangle,$$

where if we have $x = x_1 \ldots x_n$ and $w = w_1 \ldots w_n$ , then $x \cdot w = x_1 w_1 \oplus x_2 w_2 \oplus \ldots \oplus x_n w_n$.

When we apply Hadamard to $\frac{1}{\sqrt{2}} (\ket{x} + \ket{y})$ we get:

$$\frac{1}{\sqrt{2}} (\ket{x} + \ket{y}) \xrightarrow{H^n} \frac{1}{\sqrt{2^n}} \left( \sum_{w \in \{0,1\}^n} (-1)^{x \cdot w} |w\rangle + \sum_{w \in \{0,1\}^n} (-1)^{y \cdot w} |w\rangle \right) = \frac{1}{\sqrt{2^n}} \sum_{w \in \{0,1\}^n} [(-1)^{x \cdot w} + (-1)^{y \cdot w}] |w\rangle.$$

Therefore, we obtained a superposition of all inputs $w$, each with amplitude $(-1)^{x \cdot w} + (-1)^{y \cdot w}$.

Finally, we measure each of these qubits from the first register in the $(|0\rangle, |1\rangle)$ basis [12].

We notice that if $(-1)^{x \cdot w}$ and $(-1)^{y \cdot w}$ are equal, then it is possible to observe the corresponding state $w$ after the measurement.

But, if $(-1)^{x \cdot w}$ and $(-1)^{y \cdot w}$ are different, they cancel each other, so $w$ cannot be observed.

Hence, after the measurement we are going to be left with a random input $w$, such that: $(-1)^{x \cdot w} = (-1)^{y \cdot w}$. We can rewrite this relation as:

$$(-1)^{x \cdot w} = (-1)^{y \cdot w} \Leftrightarrow x \cdot w \equiv y \cdot w \mod 2,$$

which is equivalent to $(x \oplus y) \cdot w \equiv 0 \mod 2$.

But $x \oplus y = s$, thus we obtain $s \cdot w \equiv 0 \mod 2$.

Consequently, we have a random linear equation satisfied by the xor mask $s$.

We can repeat this whole process (run again Simon’s algorithm) to obtain a second linear equation:

$$\begin{aligned}
&s \cdot w_1 \equiv 0 \mod 2 \\
&s \cdot w_2 \equiv 0 \mod 2
\end{aligned}$$

In this way, if we repeat the algorithm $O(n)$ times, we will get with high probability $n$ linear equations, independent of each other and we can efficiently solve this system of linear equations [30] (for instance using Gaussian Elimination).

Then, if the system has only 1 solution $s = 0$, this means that $f$ must be bijective. Otherwise, we will obtain 2 solutions, $s_1 = 0$ and $s_2 \neq 0$, where $s_2$ represents the xor mask for $f$.

Every time we run Simon’s algorithm we query $f$ only 1 time, so in total we need $O(n)$ queries to $f$. 47
As a result, we have proved that Simon’s problem can be solved by a quantum computer in polynomial time with bounded error probability, using an oracle \( O \) which returns the values of \( f \):

\[
\text{Simon Problem } \in \text{BQP}^O
\]

7.2. Classical no-go Result

We remind that our goal is to show that \( \text{BQP} \not\subset \text{NP/poly} \cap \text{coNP/poly} \). We want to give strong evidence in favour of this relation by proving that it is true with respect to an oracle \( O \):

\[
\text{BQP}^O \not\subset \text{NP/poly}^O \cap \text{coNP/poly}^O
\]

This is the reason why we are looking for a problem which can be shown to be solvable in \( \text{BQP}^O \), but which cannot be solved in \( \text{NP/poly}^O \cap \text{coNP/poly}^O \).

The \( \text{BQP} \) class is **closed under complement** \([17]\), which means that the complement of any problem from \( \text{BQP} \) is also in \( \text{BQP} \). Therefore, the **complement of Simon’s problem** (coSimon) belongs to the \( \text{BQP} \) class. We can obtain coSimon by reversing the yes and no answers to Simon’s problem \([20]\):

The coSimon decision problem will return “yes” if a function \( f \) is a bijection and “no” if \( f \) is a 2-to-1 function.

From this point on, we will be working only with the coSimon problem. Our aim is to prove that coSimon is not in the set \( \text{NP/poly}^O \), hence it cannot be in the set \( \text{NP/poly}^O \cap \text{coNP/poly}^O \) either.

The reason why we are considering coSimon and not directly the Simon problem is the following:

Simon problem can be solved by a TM in \( \text{NP}^O \), so it also belongs to the \( \text{NP/poly}^O \) class. Therefore, we would have needed to prove that it is not in \( \text{coNP/poly}^O \). But, while the complement of \( \text{NP} \) is the class \( \text{coNP} \), \( \text{(coNP)}^O \) is **not** the complement of the class \( \text{NP}^O \). Since we do not have a definition of the class of problems represented by \( \text{(coNP)}^O \) or by \( \text{(coNP/poly)}^O \), we will instead work with the complement of Simon’s problem and show that it cannot be in \( \text{NP/poly}^O \).

7.2.1. \( \text{NP} \)-Impossibility

We begin by demonstrating the coSimon problem is not in the complexity class \( \text{NP}^O \).

**MAIN THEOREM 2.2:** Simon Problem \( \notin \text{NP}^O \).

**Proof.** Assume coSimon \( \in \text{NP}^O \). Then, there exist an Oracle Turing Machine \( M \in \text{NP} \) which accepts a function \( f \) when \( f \) is bijective and rejects \( f \) when \( f \) is 2-to-1.

As this is an oracle problem, the input for \( M \) will not be an explicit description of a function \( f \) \([21, 22]\) (for example, the value of \( f \) in every element of its domain).

Instead, we use the fact that the set of 1-to-1 and 2-to-1 functions \( f : \{0, 1\}^n \rightarrow \{0, 1\}^n \) is **countable** and thus we can associate for any such function a natural number **index**.

48
Then, $M$ will receive as input the pair $(i, n)$, where $i$ represents the index of the function and $n$ (which is written in the unary system) specifies the size of any element from the domain of the function: $(i, n)$ maps to $f_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$.

The oracle $O$ works in the following way: it receives the index $i$ of the function, the length $n$ and an element $x \in \{0, 1\}^n$ and returns to $M$ the value of $f_i(x)$ in only one computational step: $O(n, i, x) = f_i(x)$.

Suppose we would have a random oracle $O$ [22]. Then, given the length $n$, $O$ is either uniformly distributed among bijective functions or is distributed uniformly among 2-to-1 functions.

For each possible length $n$ we have an oracle $O_n$ and suppose we have $\frac{N_n}{2}$ bijections and $\frac{N_n}{2}$ 2-to-1 functions chosen for $O_n$, with $N_n = O(2^n)$.

Then we can consider our oracle $O$ as the union of all oracles $O_n \forall n \in \mathbb{N}$. $O$ has $\frac{N}{2}$ bijections and $\frac{N}{2}$ 2-to-1 functions ($N = \sum_i N_i$).

Consequently, given an index $i$ we have $P(f_i \text{ is bijective}) = P(f_i \text{ is 2-to-1}) = \frac{1}{2}$.

For instance, the oracle $O$ could be built in the following way: $\forall n \in \mathbb{N}$ and for $i \in \{1, ..., 2 \cdot N_n\}$ we randomly selected a bit $b_{n,i}$ and a string $s_{n,i}$ of length $n$. If $b_{n,i} = 1$ then the function $f_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$ chosen to be computed by oracle $O$, is 2-to-1 with the xor mask $s_{n,i}$.

If $b_{n,i} = 1$ then the function $f_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is bijective.

Instead of having a random oracle, the oracle used for our problem is adversarial [31]. This implies that $O$ does not hold any $N$ 1-to-1 or 2-to-1 functions chosen uniformly at random, but $O$ is allowed to choose $N_n$ 1-to-1 or 2-to-1 functions for any length $n$ (out of the $(2^n)!$ total number of permutations and out of the $(2^n - 1)2^n - 2^n$ total number of 2-to-1 functions).

Consequently, we need to build the oracle $O$ (select the $N_n$ functions owned by $O$) in such a way that $\text{coSimon}^O \notin NP^O$.

Our proof is based on diagonalization [16].

The set of nondeterministic $TM$ is countable [21] so we can consider an enumeration of them: $M_1, M_2, ..., M_k, ...$.

Therefore, we must construct the adversarial oracle $O$ such that:

$\forall i \in \mathbb{N} \text{ coSimon}^O \notin M_i^O$

Initially, the set of functions chosen for $O$, $\mathcal{F}$ is empty and at each step $i$ we add another function to it.

Suppose we are at step $i$ and we have the oracle nondeterministic $TM M_i^O$.

The adversarial oracle $O$ knows the computational tree of $M_i$. We can view $M_i$ as an algorithm which uses the oracle to take decisions for the paths in its computational tree. The root of the tree contains the initial configuration of $M_i$ on an input, whereas all the internal nodes are queries made to the oracle. When no more queries are made on a path, we reach a final node (a leaf), which is either an accepting or a rejecting state [20].

The input for $M_i$ consists of the index of the function and the size of any element from the function’s domain. Thus, for the input $(index, n)$, the computational tree of $M_i$ would look as seen in Figure 5:
We now describe $O$’s behaviour which must make sure that $M^O_i$ cannot always solve correctly coSimon.

The oracle picks any 1-to-1 function $g : \{0, 1\}^n \to \{0, 1\}^n$, which is correctly accepted by $M_i$. This means that if the input to $M_i$ is $(\text{index}, n)$, the oracle will place $g$ on the position index inside his function set $F$. $M_i$ accepts $g$ is equivalent to: there exists a path in the computational tree of $M_i$ on input $(\text{index}, n)$ which ends in an accepting state. This path contains $l$ strings $x_1, \ldots, x_i$ for which $M_i$ queries $O$. We call this path $\pi = x_1, x_2, \ldots, x_i$. Because $M_i$ is a nondeterministic polynomial time machine, $l$ must be polynomial in the size of input, $l = \text{poly}(n)$.

For each node $x_i \in \pi$, $M_i$ queries the oracle $O$ and obtains $g(x_i) = y_i$. Based on these results $y_i$, $M_i$ takes the decision that $\pi$ is an accepting path. In other words, we can view this as if $M_i$ has a function $F$, which encodes its decision algorithm. The only things $M$ knows after reaching the end of $\pi$ are: the input he received at the beginning and the oracle’s answers $(y_1, y_2, \ldots, y_l)$. Therefore, $F$’s output can only depend on these values, so we have that $F(y_1, y_2, \ldots, y_l, \text{index}, n) = 1$.

Now, what $O$ basically found, is that given the input $(\text{index}, n)$ and a series of strings $y_i, i \in \{1, \ldots, l\}$, received by $M_i$ in response for the queries for some inputs $x_i, i \in \{1, \ldots, l\}$, the path $\pi$ will be an accepting path and therefore $M_i$ will accept the function on the position index.

The adversarial oracle $O$ wants to force $M_i$ to give a wrong answer. All $O$ needs to do is find a 2-to-1 function $\hat{f} : \{0, 1\}^n \to \{0, 1\}^n$ which will be placed on position index in $F$ (replace the function $g$, which was initially set on that position) and which must satisfy the following conditions:

$$\begin{align*}
\hat{f}(x_1) &= y_1 \\
\hat{f}(x_2) &= y_2 \\
&\vdots \\
\hat{f}(x_l) &= y_l
\end{align*}$$

(C$_1$)

In this way, when $M^O_i$ runs the input $(\text{index}, n)$ for the function $\hat{f}$, the path $\pi$ will end in an accepting state because the machine gets exactly the same answers as for the function $g$ which was accepted by $\pi$. Thus, $M^O_i$ will decide that $\hat{f}$ is bijective, which is false.

At this point we need to show there exists such a function $\hat{f}$ satisfying (C$_1$) and we need to properly define it. All the inputs along any path of the computation tree of a nondeterministic machine must be different, thus $(x_1, x_2, \ldots, x_i)$ must all be distinct. Because $g$ is bijective this means that $g(x_1)$, $g(x_2)$, $\ldots$, $g(x_i)$ are also distinct.
Therefore, we need to make sure that for the 2-to-1 function \( \hat{f} \), there can be no pair \((p, q)\), with \(1 \leq p < q \leq l\), such that \( \hat{f}(x_p) = \hat{f}(x_q) \).

Hence, we must also set some constraints about the associated xor mask string \( s' \) of function \( \hat{f} \).

We know that \( \hat{f}(x_p) = \hat{f}(x_q) \) if and only if \( s' = x_p \oplus x_q \).

This means that we must consider all the possible pairs of inputs from path \( \pi \), \((x_p, x_q)\) and make sure that \( s' \) is different from \( x_p \oplus x_q \).

The number of all possible distinct strings \( c_{p, q} = x_p \oplus x_q \) is at most \( \binom{l}{2} = \frac{(l-1)l}{2} \).

\( M_i \) is a nondeterministic polynomial-time TM, so every path in the computational tree contains a polynomial in \( n \) number of queries, \( l \leq p(n) \). Therefore, \( \frac{(l-1)l}{2} < 2^n \).

We can choose \( s_k \) be any of the \( 2^n \) strings in \( \{0, 1\}^n \) except those \( \frac{(l-1)l}{2} \) \( c_{p, q} \) strings.

(\( C_2 \))

Now, since \( \hat{f} \) is 2-to-1 with the xor mask \( s' \) we have \( \hat{f}(x') = \hat{f}(x) \) if and only if \( x' = x \oplus s' \).

Thus, after we satisfy condition \( C_2 \) we will have all \( \hat{f}(x_i) \) distinct from each other, \( 1 \leq i \leq l \).

When defining a 2-to-1 function \( h : \{0, 1\}^n \rightarrow \{0, 1\}^n \) given a xor mask \( s \), it suffices to set the values for half of the inputs \((x_{i_1}, x_{i_2}, \ldots, x_{i_{2^{n-1}}})\) such that: \( \forall 1 \leq a, b \leq 2^{n-1}, x_{i_a} \neq x_{i_b} \oplus s \) and \( h(x_{i_a}) \neq h(x_{i_b}) \). For the other half of the inputs, the values of \( h \) are set by the relation:

\[ h(x_{i_j} \oplus s) = h(x_{i_j}) \quad \forall 1 \leq j \leq 2^{n-1}. \]

Therefore, in order to define \( \hat{f} \), we can set the values \( \hat{f}(x_1) = \hat{f}(x_1 \oplus s') = y_1 \), ..., \( f''(x_i) = \hat{f}(x_i \oplus s') = y_i \), whereas for the rest of the inputs we can pair them together \((x_{i_j}, x_{i_j} \oplus s')\) and assign to both of them a value from \( \{0, 1\}^n \) which has not been given to any of the previously assigned inputs.

Consequently, we have constructed a 2-to-1 function \( \hat{f} \) satisfying \( C_1 \) and we know that \( M_i \) will give a wrong answer for \( \hat{f} \) (by accepting it).

We repeat the same procedure for all nondeterministic TM \( M_i \), obtaining that for all of them we can define a 2-to-1 function which the machines will mistakenly identify as 1-to-1.

\textbf{OBS 8.} Another aspect we need to take into consideration is that the index specified in the input used to deceive the Turing Machine \( M_i \), should be different from the previously selected indices used to deceive the Turing Machines \( M_1, M_2, \ldots, M_{i-1} \).

The reason is that if the input to \( M_i \) maps to an already set position in \( F \), occupied by some 2-to-1 function \( f'' \) used to trick the machine \( M_k, k < i \), then we will replace \( f' \) by a new function \( f''' \) such that \( M_i \) gives the wrong answer for \( f''' \). However, it can be the case that \( f''' \) may now be correctly rejected by \( M_k \). Thus, we wouldn’t have any more that \( M_k \) cannot decide coSimon.

In order to avoid this problem, at each step \( i \), we run \( M_i \) on the input with index \( i \), and hence we add in \( F \) the resulting 2-to-1 function on position \( i \).

Consequently, coSimon\(^0 \not\in M_i \) \( \forall i \in \mathbb{N}, M_i \in NP \), or in other words, coSimon\(^0 \not\in NP \). □
7.2.2. \textit{NP/poly}-Impossibility

The next step is proving that coSimon cannot be solved by a nondeterministic polynomial time algorithm receiving help from a polynomial advice:

\textbf{MAIN THEOREM 2.3:} \textit{coSimon} \notin \textit{NP/poly}^0.

\textbf{Proof.} In order to show this, we need to indicate how the adversarial oracle \( O \) can deceive any nondeterministic TM \( M_i \), irrespective of what advice string the machine might receive.

Again, we proceed with a proof by \textit{contradiction}.

Suppose \textit{coSimon} \( \in \textit{NP/poly}^0 \).

Then, there exists an Oracle Turing Machine \( M \) in \( \textit{NP/poly} \) which returns “yes” when \( f \) is bijective and “no” when \( f \) is 2-to-1.

As opposed to the \( \textit{NP} \) case, now \( M \) also receives some nonuniform information [18] to help decide if a function is bijective or not. This information is given by an advice function, \( h : \mathbb{N} \to \Sigma^* \), such that \( |h(n)| \leq p(n) \) \( \forall n \), \( p \) a polynomial. However, this advice is the same for all inputs of \( M \) which have the same length [20, 24].

The input for \( M \) is represented by \( i \) the index of a function, which is a natural number between 1 and \( N_n \) and \( n \) the size of the inputs for that function, with \( n \) represented in the unary system. This means that for any function \( f_i : \{0, 1\}^n \to \{0, 1\}^m \) (1-to-1 or 2-to-1) we have the same string advice \( s_{\text{advice}} \) such that \( |s_{\text{advice}}| \leq p(m + \log(\text{number of functions given } m)) \).

Suppose the number of functions the oracle holds for any length \( m \) is \( N_m = 2^{q(m)} \), \( q \) polynomial. Then, \( |s_{\text{advice}}| \leq p(m + q(m)) \), so \( |s_{\text{advice}}| = O(p(q(m))) \).

We first present the intuition of the proof by \textit{contradiction}.

\textbf{Intuitive proof.} The number of 2-to-1 and 1-to-1 functions defined on the set \( \{0, 1\}^n \) is \( N_n = 2^{q(n)} \) which is exponential, but we receive the same advice of length polynomial in \( n \), for all these functions. The advice could help the nondeterministic machine by giving for each of these \( N_n \) functions some information which can be used in the process of decision.

However, we have the same external information of length \( p(q(n)) \) for any of the \( 2^{q(n)} \) functions [18]. Because \( p(q(n)) \leq 2^{q(n)} \) \( \forall p, q \) polynomials this would mean that the advice \textbf{cannot even provide one single bit of information} for each of these functions, necessary to \textbf{distinguish} between them.

If instead, the size of the advice would have been equal to \( N_n \), then we would have 1 bit of information assigned for each function. In this case, the advice string could have indicated directly the solution: each bit of advice \( b_i \) would be \( b_i = 0 \) if the function \( f_i \) is 2-to-1 and \( b_i = 1 \) if \( f_i \) is bijective and thus the machine could have determined the type of any function.

But in general, if the number of functions is larger than the number of bits of the advice shared between all functions, we cannot obtain any kind of information for each of these functions.

Despite this, the advice string could provide external information [28] for a polynomial number of functions. For example, considering the first \( p(q(n)) \) functions, the advice could provide one bit of information for each of these functions and therefore, \( M \) could decide for them if they are bijective or not. Nonetheless, for the rest of the functions, the \( \textit{NP} \) machine would receive no additional external help to decide whether to accept them or reject.

Therefore, for any length of the input \( n \), we would have \( 2^{q(n)} - p(q(n)) \) functions for which the \( \textit{NP} \) machine \( M \) would have no extra information and thus the proof \textbf{reduces} to the case of coSimon \( \notin \textit{NP}^{\text{poly}} \) discussed before. \( \square \)
We will next present a more formal evidence for the MAIN THEOREM 2.3. 

**Formal proof.** Suppose there exists a \( NP/\text{poly}^0 \) \( TM \) \( M_i \) solving the cosimon problem. The input for \( M_i \) is a string \( ldx \) encoding the index of the function and the length of the input for that function \( (ldx = (\text{index}, n)) \).

Suppose from now on we will work only with a fixed length \( n \) (the results we will obtain can then be generalised for all possible lengths). The maximum index of a function is a number exponential in \( n \), \( [\text{index}] = q(n) \), \( q \) polynomial. Therefore, we obtain that the size of the input for \( M_i \) is \( O(q(n)) \). In this situation, the advice received by \( M_i \) for all functions has the property that \( |\text{advice}| = p(q(n)) \), \( p \) polynomial. As \( n \) is fixed, we can assume that \( ldx \) only refers to a function index in \( \mathcal{F} \).

To build our proof we will start from some simplified problems.

**Problem 1.** Let’s first assume \( M_i \) decides the type of the function associated with the input \( ldx \), only based on the advice string and on \( ldx \). In other words, \( M_i \) makes no queries to the oracle and the decision procedure for a path can be viewed as: \( F(ldx, \text{advice}) \).

**Solution.** We can view the problem in this way: we have no control upon what the advice string might be, but we do control what function to associate for an input \( ldx \) such that we deceive \( M_i \).

We build a matrix \( \text{Mat} \) (Figure 6), where each row represents a possible advice which \( M_i \) might receive and each column represents a possible input to \( M_i \). Each cell in \( \text{Mat} \), \( \text{Mat}_{j,k} \) contains a binary value, specifying whether \( M_i \), by receiving the advice \( \text{adv}_j \), says that the function on position \( k \) is bijective \( (\text{Mat}_{j,k} = 1) \) or 2-to-1 \( (\text{Mat}_{j,k} = 0) \).

![Figure 6. Example of matrix recording the decisions made by \( M_i \), depending on the advice received](image)

For every input \( ldx \) the oracle \( O \) chooses a function which maximizes the number of wrong answers given by \( M_i^0 \).

For the first column, we have \( 2^p(q(n)) \) responses, 0 or 1, depending on the advice received by \( M_i \). We count the number of 0 (“reject”) and 1 (“accept”) responses gave by \( M_i \) and we obtain \( u \) “accept” answers and \( v \) “reject” responses, such that \( u + v = 2^{p(q(n))} \).

If \( u > v \), then for the first position in the set of functions the oracle selects a 2-to-1 function. Otherwise, the oracle selects a 1-to-1 function. In this way, for more than a half of the total number of advices, \( M_i \) gave the wrong answer, so we can rule out these advices. Then, out of the total \( 2^{p(q(n))} \) advices we now look at only \( \min(u, v) \) number of advices for the next column, where \( \min(u, v) \) is obviously less than \( \frac{2^{p(q(n))}}{2} \).
And at the second column, we proceed in a similar manner: we look at the responses $M_i$ gave for the remaining advices and consider a function of the other type than the type specified by the majority of answers. This function will be placed on the second position. The number of advices left is now less than $\frac{2^{p(q(n))}}{2^k}$. In this manner, for every column we reduce with more than a half the number of advices which could be correct.

Therefore, at the $p(q(n)) + 1$ column of Mat we will definitely be left with no correct advice and we conclude that there is no polynomial advice which could help $M_i$ solve this problem.

**Problem 2.** Now, we move to a second problem, specifically, showing that coSimon $\not\in P/poly$.

**Solution.** We use a similar demonstration based on the construction of a matrix recording all the possibilities for the behaviour of a $P/poly$ $TM$ $M_i$ which would be able to solve coSimon. For this case when $M_i$ is a deterministic polynomial time machine, we know that it can only query the oracle $O$ for a polynomial number of times [17, 20].

Suppose $M_i$ queries $O$ at most $k$ times ($k$ polynomial in $n$), with all the inputs queried distinct from one another. These queries can be any strings of length $n$, and we name them: $x_1, x_2, \ldots, x_k$.

We build a new matrix $\text{Mat}'$ (Figure 7). The rows of $\text{Mat}'$ represent the possible advices which $M_i$ could have received and the possible $k$ inputs queried by $M_i$. The columns represent the function indices and the answers given by $O$ in response to the $k$ queries. The values in each cell of $\text{Mat}'$ specify whether $M_i$ receiving a particular advice and a particular set of responses from the oracle, accepts or not the function positioned at index.

If $M_i$ would have not received any external advice, the oracle $O$ would know its algorithm and so would know the queries the machine is going to make [22]. However, an advice can change the behaviour of $M_i$ and determine a new sequence of queries requested by $M_i$.

Thus, for every possible external information, we have to consider a new behaviour of $M_i$, each of them can be seen as a new different deterministic polynomial time $TM$ [21]. Since each such deterministic polynomial-time $TM$ is identified by a sequence of at most $k$ queries, we only need to take into account every possible array of $k$ different queries.

Hence, each advice $adv_i$ corresponds to a sequence $(x_{i,1}, x_{i,2}, \ldots, x_{i,k})$ (which we call $k$-sequence), $x_{i,j} \in \{0,1\}^n$ $\forall j \in \{1,\ldots,k\}$, $\forall i \in \{1,\ldots,2^{p(q(n))}\}$. We name $k$-answer, the array of the $k$ answers given by the oracle in response to a $k$-sequence. Our goal is that, for every function index, to eliminate a fraction of these possible $k$-sequences such that, for the last index we will be left with no possible $k$-sequence. Then, we conclude that no advice can help a $P$ machine solve the coSimon problem.

<table>
<thead>
<tr>
<th>advice/ $k$-queries</th>
<th>Index function query answers</th>
<th>Index function = 1</th>
<th>...</th>
<th>Index function = $2^{q(n)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(x_{1,1}, \ldots, x_{1,k})$</td>
<td>$(x_{1,1}, \ldots, x_{1,k})$</td>
<td>...</td>
<td>$(x_{1,1}, \ldots, x_{1,k})$</td>
</tr>
<tr>
<td>$adv_1$</td>
<td>$(x_{1,1}, \ldots, x_{1,k})$</td>
<td>$1$</td>
<td>...</td>
<td>$0$</td>
</tr>
<tr>
<td>$adv_2$</td>
<td>$(x_{2,1}, \ldots, x_{2,k})$</td>
<td>$0$</td>
<td>...</td>
<td>$0$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$adv_{2^{p(q(n))}}$</td>
<td>$(x_{2^{p(q(n))},1}, \ldots, x_{2^{p(q(n))},k})$</td>
<td>$-$</td>
<td>...</td>
<td>$1$</td>
</tr>
</tbody>
</table>

*Figure 7: Example of matrix for the P/poly case*
Every response to a query is a number between 0 and $2^n - 1$. Thus, the number of possible $k$-answers (containing $k$ distinct answers) given in response to a $k$-sequence asked by $M_i$ is equal to the number of arrangements of $2^n$ answers taken $k$ at a time, $P(2^n, k)$. However, the function may be 2-to-1, so a $k$-answer must not always contain $k$ distinct answers (we need to also allow for some pairs from the $k$ values to be equal). We could consider any possible xor mask for $f$ and determine for which inputs $(x_i, x_j)$ we can have $f(x_i) = f(x_j)$. One solution might be listing all possible $2^n$ values for the elements of a $k$-answer and then we invalidate the entries in the matrix which cannot be a subset of the set of values of any bijective or 2-to-1 function. For example, we could insert in those invalid corresponding cells the symbol “-”.

Therefore, the number of columns of $Mat'$ is $V$, where $2^{q(n)} \cdot P(2^n, k) < V < 2^{q(n)} \cdot 2^{nk}$.

Unlike the previous problem, for each function index we now have a block of columns (instead of just one column) and every column represents a possible $k$-answer.

Consequently, for function $index = i$ we initially set a 1-to-1 function $f$. We then look at the results provided by $M_i$ (accept/reject) given each of the advices available at this stage. We count the total number of accepts and rejects. If the number of rejects is greater than the number of accepts, then it means that more than a half of the advices are wrong and we can eliminate them and move to the next step (function index $i + 1$).

Otherwise, we can proceed in the following way. We select half of the inputs, for instance the first $2^{n-1}$ inputs, $S = \{0, 1, ..., 2^{n-1} - 1\}$ and construct a 2-to-1 function $g$, such that $g(0) = f(0), g(1) = f(1), ..., g(2^{n-1} - 1) = f(2^{n-1} - 1)$.

We can build such a function $g$, by setting its associated xor mask $s_g$ such that $s_g \neq u \oplus v \forall u, v \in \{0, ..., 2^{n-1} - 1\}, u \neq v$. Such a string $s_g$ exists, because we notice that all the values of $u \oplus v$ are less than $2^{n-1} - 1$, so, for example, we can set $s_g = 2^{n-1}$.

We now consider the majority of advices $adv_j$, the ones for which the resulting deterministic polynomial time TMs are saying that the function with the index $i$ is bijective. Each of these TMs corresponds to a $k$-sequence and suppose the number of these machines is $L$ (we know that $L > \frac{1}{2} \cdot \text{number of possible advices}$). Now, in the worst case, each advice $adv_j$ corresponds to a different $k$-sequence, meaning that $L = \binom{2^n}{k}$. This situation is possible because we cannot control the exact size of the advice, we just know that the external advice must be polynomial in the size of input. Thus, in the worst case, we can have $|\text{advice}| = \log \left( \binom{2^n}{k} \right)$, which is a polynomial function in $n$.

By selecting only the deterministic TMs whose queries consist of inputs from the set $S$, we have that all these TMs will accept function $g$, as they get the same query responses as for the $f$ function. But, since $g$ is 2-to-1, they are all wrong and therefore, we can eliminate the lines of the matrix corresponding to these advices. The number of $k$-sequences consisting of inputs only from the set $S$ is $\binom{2^{n-1}}{k}$, so at each step we can eliminate at least $\binom{2^{n-1}}{k}$ lines of the matrix $Mat'$.

We can further improve the number of eliminations by making the next essential observation.

**OBS 8.** We can additionally eliminate $k$-sequences which also contain inputs $x'$ from the other half of inputs $(\{0, 1\}^n - S)$ as long as $x' \neq x \oplus s_g \forall x \in S$. This is because we are assuming that each of the remaining TMs can solve the coSimon problem, therefore, they always accept correctly a bijective function. In other words, if the oracle returns any $k$ values which could be the values of any bijective function (not only from $f$), then these TMs will accept.
Thus, the only condition the oracle must satisfy in order to deceive a deterministic \( TM \), is that for \( k \) distinct inputs to return \( k \) distinct values. For this reason, we now take into account any \( k \) sequence for which the fixed 2-to-1 function \( g \) returns distinct values.

Therefore, the number of \( k \)-sequences we can select is: 

\[
E = \binom{2^{n-1}}{0} \cdot \binom{2^{n-1}}{k} + \binom{2^{n-1}}{1} \cdot \binom{2^{n-1}}{k-1} + \cdots + \binom{2^{n-1}}{k}. \]

We further obtain that 

\[
E = \sum_{i=0}^{k} \binom{2^{n-1}}{i} \cdot \binom{2^{n-1}-i}{k-i} = \frac{(2^{n-1})!}{k!} \cdot \frac{1}{k!} \sum_{i=0}^{k} \frac{(2^{n-1})!}{(2^{n-1}-k)!} \cdot \binom{2^{n-1}}{k} = \frac{2^{n-1}!}{k!} \cdot \frac{1}{k!} \cdot 2^k = \frac{(2^{n-1})!}{(2^{n-1}-k)!} \cdot \frac{1}{k!} \cdot 2^k.
\]

For all these \( E \) queries the resulting \( TM \)s will give the wrong answer (saying that \( g \) is bijective), so we can eliminate \( E \) lines of the matrix \( Mat' \) in one step.

We can see that we are now left with queries containing inputs both from \( S \) and from \( \{0, 1\}^n - S \), such that we have at least one pair \((x, x')\), where \( x \in S, x' \in \{0, 1\}^n - S \) and \( x' = x \oplus s \).

Then, we can repeat the whole procedure for all \( 2^{q(n)} \) function indices. We obtain that we are able to eliminate in the worst case at least \( 2^{q(n)} \cdot \binom{2^{n-1}}{k} \cdot 2^k \) advices. As \( 2^{q(n)} \cdot \binom{2^{n-1}}{k} \cdot 2^k \geq \binom{2^n}{k} \), this means that we have discarded all lines of \( Mat' \), so no possible polynomial advice could help any deterministic machine solve \( coSimon \). Therefore, \( coSimon \notin \text{P/poly} \).

Using the proof techniques used in \textbf{Problem 1} and \textbf{Problem 2}, we can finally proceed to demonstrate \textbf{MAIN THEOREM 2.3}.

Suppose \( 3M_i \in \text{NP/poly} \) solving \( coSimon \).

Once again, we need to take into consideration every possible advice received by \( M_i \). As in the previous problems, the number of advices is \( 2^{p(q(n))} \), where \( p \) and \( q \) are 2 polynomials.

For each of these possible advice strings which \( M_i \) might receive, we obtain a new nondeterministic polynomial-time machine \( M_{ij} \) \cite{17, 20} and we must demonstrate that none of these resulting machines is able to solve our \( coSimon \).

Following the same strategy, we build a new matrix \( Mat'' \) (Figure 8), where each line corresponds to a possible advice (the resulting \( NP \) machine) and for each possible function index we have a block of columns corresponding to possible \( k \)-answers returned by the oracle \( O \).

![Figure 8. Example of matrix for the P/poly case](image-url)
For the first step (function \(i\)ndex \(= 1\)), we choose a bijective function \(f \colon \{0, 1\}^n \to \{0, 1\}^n\).

In this way, we fixed the answers the oracle will return to each \(M_{i,j}\). This results in a single column for \(i\)ndex \(= 1\), no matter what the computational trees of each \(M_{i,j}\) would look like. Each of the values in this column will be a 0 or 1 specifying if \(M_{i,j}\) accepts or not this \(f\). We then iterate through all possible advices and look at the answers gave by each machine \(M_{i,j}\).

We can count the number of accepts and rejects. If the number of rejects is larger, this means that the more than a half of the TMs are wrong and we can eliminate this majority for the following step.

Now, consider the case when the number of “yes” answers is in majority. Suppose the number of these machines is \(L\), \(L > \frac{2^{p(q(n))}}{2}\). Then, we want to show that we can eliminate at each stage a fraction of these \(L\) lines, such that at the final step index \(= 2^{q(n)}\), we are left with no possible nondeterministic machines capable of solving our problem. If we can eliminate at each step a constant fraction of lines \(c\), then at step \(i\) we would have \(2^{p(q(n))} \cdot (1 - c)^i\) remaining lines and until we reach the last index function we would eliminate all advices.

From the selected 1-to-1 function \(f\) we want to build a 2-to-1 function \(\hat{f}\) in the following manner:

We choose the first \(2^{n-1}\) inputs \(S = \{0, 1, \ldots, 2^{n-1}\}\) and construct a 2-to-1 function \(\hat{f}\) with the xor mask \(s\), such that \(\hat{f}(0) = f(0)\), \(\hat{f}(1) = f(1)\), \(\ldots\), \(\hat{f}(2^{n-1}) = f(2^{n-1})\) and \(s \neq u \oplus v\) \(\forall u, v \in \{0, \ldots, 2^{n-1} - 1\}\), \(u \neq v\) (for instance \(s = 2^{n-1}\)).

Each of the \(L\) machines is incorrectly accepting \(f\). Because they are nondeterministic polynomial-time machines, this fact is equivalent to saying that: for any of these TMs, there exists a path containing a polynomial number of queries, such that the path ends in an accepting state [20, 33].

Therefore, for each advice \(adj\) we can associate an accepting path \(\pi_j\) of the machine \(M_{i,j}\). We suppose each of these paths contains a sequence of at most \(k\) queries to the oracle, where \(k\) is polynomial in \(n\). Hence, if we assume that for a given advice, the correspondent machine \(M_{i,j}\) always accepts correctly a bijective function, then by giving the answers to the \(k\)-sequence as coming from any bijective function, \(M_{i,j}\) must accept the given function.

We pick an accepting path for each of these nondeterministic machines. Then, we can deceive a path \(\pi_j\) by returning in response to the queries along \(\pi_j\), a \(k\)-answer consisting of distinct values coming from a 2-to-1 function. We can see each of these accepting paths \(\pi_j\) as a deterministic polynomial time machine. Then, we deduce that we only need to consider all possible \(k\)-sequences and eliminate them until the last step (function index \(= 2^{q(n)}\)). Consequently, the proof reduces to the Problem 2 of showing that no \(P/poly\) machine can solve the \(coSimon\) problem, so we conclude that:

\[
coSimon \notin NP/poly
\]

In this section we have described the problem \(coSimon\) solvable by a BQP machine and we presented both an intuitive and a formal proof that \(coSimon \notin NP/poly^0\). Using the fact that \(coSimon \in BQP^0\) and that any problem in \(BQP^0\) can be reduced in polynomial time to a problem \(G\) from \(BQP - Hard^O\) [17], we can derive that no \(BQP - Hard^O\) problem can be solved by a machine from the class \(NP/poly^0\).

In the next part, we will present and analyse another problem named Forrelation [25], which suggests the gap between the complexity classes \(BQP\) and \(NP/poly\).
8. Forrelation Problem

Another interesting problem we study in order to indicate the separation between the $BQP$ and the $NP/poly$ classes is called the Forrelation Problem [25]. One reason why we delve into this problem is that it proves the highest gap known so far between classical and quantum complexity.

DEFINITION 9: The Forrelation problem determines whether one Boolean function $f : \{0, 1\}^n \to \{-1, 1\}$ is highly correlated with the Fourier transform $\tilde{g}$ of a second Boolean function $g : \{0, 1\}^n \to \{-1, 1\}$, given the promise that $f$ and $\tilde{g}$ are either highly correlated or strongly uncorrelated.

Specifically, the level of correlation between $f$ and $\tilde{g}$ is measured by the quantity:

$$\Theta_{f,\tilde{g}} = \frac{1}{2^n} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} f(x) g(y).$$

The Forrelation problem tells us that either $|\Theta_{f,\tilde{g}}| < \frac{1}{100}$ - corresponding to a high correlation or $\Theta_{f,\tilde{g}} \geq \frac{3}{5}$ - corresponding to a large uncorrelation.

As in the case of Simon’s problem, we do not have direct access to the table values of the 2 functions, but we have an oracle $O$ which given an input can return the values of the 2 functions in that point.

While classically we would need at least $\frac{2^{n-1}}{n}$ queries to solve the problem [25, 30], we will demonstrate that in the quantum paradigm Forrelation can be solved using only 1 query to the oracle.

8.1. Quantum Algorithm

**THEOREM 3.1.** Forrelation Problem $\in BQP^O$

**Proof.** The Forrelation problem can be solved using the quantum circuit in Figure 9:

![Figure 9. Circuit for solving the Forrelation problem](image)

The initial state is $|\psi_0\rangle = |0\rangle^\otimes n$. After we apply $n$ Hadamard gates we reach the state [1]:

$$|\psi_1\rangle = \frac{1}{(\sqrt{2})^n} \cdot (|0\rangle + |1\rangle)^n = \frac{1}{(\sqrt{2})^n} \cdot \sum_{x \in \{0,1\}^n} |x\rangle.$$
Next, we execute the unitary for the $f$ function [29], which has the following effect:

$$|x\rangle \mapsto U_f \cdot f(x) \cdot |x\rangle$$

Therefore, we obtain $|\psi_2\rangle = \frac{1}{(\sqrt{2})^n} \cdot \sum_{x \in \{0,1\}^n} f(x) \cdot |x\rangle$

The second round of Hadamards, will transform this state to:

$$|\psi_3\rangle = \frac{1}{(\sqrt{2})^n} \cdot \frac{1}{(\sqrt{2})^n} \cdot \sum_{x \in \{0,1\}^n} \left( \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} \cdot f(x) \cdot |y\rangle \right)$$

Following this, we apply the second unitary for the $g$ function, $U_g$, obtaining the state $|\psi_4\rangle$:

$$|\psi_4\rangle = \frac{1}{2^n} \cdot \sum_{x \in \{0,1\}^n} \left( \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} \cdot f(x) \cdot f(y) \cdot |y\rangle \right)$$

Finally, the last set of Hadamard gates will lead us to the state $|\psi_5\rangle$:

$$|\psi_5\rangle = \frac{1}{2^n} \cdot \frac{1}{2^n} \cdot \sum_{x \in \{0,1\}^n} \left( \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} \cdot f(x) \cdot g(y) \cdot |z\rangle \right)$$

Now, we measure the probability of getting $|0\rangle^\otimes n$ [12].

Using the projector $P_0 = |0\rangle^\otimes n \langle 0|^\otimes n$, we obtain the probability equals to $P(0) = \langle \psi_5 | P_0 + P_0 | \psi_5 \rangle$

$$P(0) = \left[ \frac{1}{2^n} \cdot \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} f(x) g(y) \right] \cdot \left[ \frac{1}{2^n} \cdot \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} f(x) g(y) \right]$$

The amplitude associated with the state $|0\rangle^\otimes n$ is $A = \sqrt{P(0)} = \frac{1}{2^n} \cdot \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} f(x) g(y)$, which is exactly the quantity $\theta_{f,g}$ we needed.

Moreover, we observe that we obtained $\theta_{f,g}$ with a single query to the oracle for $f$ and a single query to the oracle for $g$. Consequently, we have described a $BQP$ algorithm solving the Forrelation problem.

$$\square$$

8.2. $k$-Forrelation

We create a generalization of this problem, called $k$-Forrelation [25] which involves $k$ Boolean functions. We choose to discuss about this problem because it has the important property of being a $BQP - complete$ problem [17, 19].

**DEFINITION 10.** In the $k$-Forrelation we have the functions $f_1, f_2, ..., f_k$ with $f_i: \{0,1\}^n \rightarrow \{-1, 1\}$, $\forall i \in \{1, ..., k\}$. We compute the value of:

$$\theta_{f_1, f_2, ..., f_k} = \frac{1}{(k+1)n} \cdot \sum_{x_1, x_2, ..., x_k \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdot (-1)^{x_2 \cdot x_3} \cdot \ldots \cdot (-1)^{x_{k-1} \cdot x_k} \cdot f_1(x_1) \cdot \ldots \cdot f_k(x_k)$$

and need to decide whether $\theta_{f_1, f_2, ..., f_k} \geq \frac{3}{5}$ or $|\theta_{f_1, f_2, ..., f_k}| \leq \frac{1}{100}$.

To prove that $k$-Forrelation is a $BQP - complete$ problem, we need to show that it is both a $BQP$ and a $BQP - hard$ problem [20].
THEOREM 3.1.1: \( k \)-Forrelation \( \in BQP^O \)

Proof. We can follow a generalisation of the proof for THEOREM 3.1 (Fig. 10).

![Circuit Diagram](image)

As in the case of the Forrelation problem, the value of quantity we are interested in, \( \Theta_{f_1, f_2, \ldots, f_k} \), is represented by the amplitude corresponding to the state \(|0\rangle^n \) from the final superposition quantum state resulted from the above \( C_{\text{FORR}} \) circuit [25].

We can see that we only need to make one query to the oracles for each of the \( k \) functions, so the \( k \)-Forrelation problem can be solved using \( k \) quantum queries. \( \square \)

THEOREM 3.1.2: \( k \)-Forrelation \( \in BQP \) – Hard\(^O \).

Proof. In order to show that it is a \( BQP \) – hard problem, we use a polynomial-time reduction [32] from a problem known to be in \( BQP \) – complete: the QSIM problem [30]. In QSIM we receive as input a quantum circuit \( Q \) containing only Hadamard and CSIGN quantum gates. The problem asks us to decide whether the probability of \( Q \) ending in the state \(|0\rangle^n \) resides in one of \( 2 \) possible intervals. To be more specific, given that for amplitude corresponding to this probability \( S_Q = (0|\otimes^n Q|0)\otimes^n \) we have that either \( |S_Q| \leq \frac{1}{100} \) or \( S_Q \geq \frac{3}{5} \), the QSIM problem requires us to identify which of the 2 cases holds for the quantum circuit \( Q \).

We notice that both \( S_Q \) in the QSIM problem and \( \Theta_{f_1, f_2, \ldots, f_k} \) in the \( k \)-Forrelation are promised to be in the exact same intervals [25, 30]. Therefore, a reduction from QSIM to \( k \)-Forrelation entails showing that for any circuit \( Q \) consisting of Hadamard or CSIGN gates we are able to find \( k \) Boolean functions \( f_1, f_2, \ldots, f_k \) such that \( S_Q = \Theta_{f_1, f_2, \ldots, f_k} \).

As seen in Figure 10, \( \Theta_{f_1, f_2, \ldots, f_k} \) is obtained from a circuit \( C_{\text{FORR}} \) which contains \( k \) blocks of gates, where each block has a Hadamard gate applied on every qubit, followed by a unitary operator for a function.

Our reduction implies that this circuit \( C_{\text{FORR}} \) must match any possible circuit consisting of Hadamard and CSIGN gates only. \( C_{\text{FORR}} \) already contains Hadamard gates, thus, we must use the function operators \( U_{f_i} \) to simulate CSIGN gates [1].

We remind that the CSIGN unitary takes as input 3 qubits \(|x\rangle|y\rangle|z\rangle\) and returns \((-1)^{x\cdot y\cdot z}|x\rangle|y\rangle|z\rangle\).

Suppose \( Q \) receives an \( n \) qubits input and contains a CSIGN gate applied on the qubits \( x_i, x_j \) and \( x_k \). Then, we can simulate \( Q \) by selecting one of our \( k \) functions to be \( f_i(x_1, x_2, \ldots, x_n) = (-1)^{x_i \cdot x_j \cdot x_k} \).

Therefore, \( f_i \) affects only the 3 qubits and leaves the rest of them unchanged.
However, as seen in $C_{\text{FOR}}$, any unitary $U_f$ is also surrounded by $n$ Hadamards to the left and $n$ to the right. If we just want to apply a CCSIGN on 3 qubits and have no Hadamards applied before or after, then we can set the functions $f_{i-1}$ and $f_{i+1}$ to be constant: $f_{i-1} = f_{i+1} = 1$. If $f_{i-1} = 1$, then for $f_{i-1}$ we have: a block containing $n$ Hadamards followed by the unitary for $f_{i-1}$, which is now the identity matrix, followed by another $n$ Hadamards. Thus, the structure for $f_{i-1}$ reduces to a pair of Hadamards applied to each qubit. But since we know that $H^2 = I$ [29], all these Hadamards cancel each other. The same process takes place for $f_{i+1}$, so we are only left with the CCSIGN gate obtained by $f_i$.

So far we treated the cases when we apply Hadamards on all $n$ qubits or we cancel all Hadamards. Now, we must find a solution when in $Q$ there are $H$ gates applied only to a subset of the $n$ qubits.

Suppose we want to apply a Hadamard to only 2 qubits $x_i, x_j$, without affecting the rest of the qubits.

In this case we add 3 functions: $f_i, f_{i+1}, f_{i+2}$, such that:

$$f_i(x_1, x_2, ..., x_n) = f_{i+1}(x_1, x_2, ..., x_n) = f_{i+2}(x_1, x_2, ..., x_n) = (-1)^{x_i \cdot x_j}$$

Then, this would result in the following fragment of the $C_{\text{FOR}}$ circuit [25] (Figure 11).

![Figure 11. Fragment of Forrelation circuit to simulate Hadamard on 2 qubits](image)

The result of this circuit is the application on the qubits $x_i, x_j$ of the operator:

$$U = H_2 \cdot \text{CSIGN} \cdot H_2 \cdot \text{CSIGN} \cdot H_2 \cdot \text{CSIGN} \cdot H_2$$

We use the relation that $(H_2 \cdot \text{CSIGN})^3 = \text{SWAP}$ [3, 12]. Therefore, we have $U = \text{SWAP} \cdot H_2$.

On the other hand, for the other $n - 2$ qubits (different from $x_i, x_j$), the 3 functions $f_i, f_{i+1}, f_{i+2}$ do not alter them, so we only apply the Hadamards on those $n - 2$ qubits, resulting in the following operator acting on them:

$$U' = H \cdot I \cdot H \cdot I \cdot H \cdot I \cdot H = H^4 = I,$$

which means that the other $n - 2$ qubits remain unchanged as we wanted.

However, we saw that the operator $U$ acting on $x_i, x_j$ has besides the Hadamard, an additional operator, SWAP. But, since we are aware of this fact, all we need to do is to swap the 2 qubits $x_i, x_j$ every time we want to apply a Hadamard on them.

Now, we can generalize our result for the application of Hadamard gates on a subset of qubits, of size larger than 2. Consider we want $H$ gates acting on $m$ qubits. Then, if $m$ is even we can group these qubits into $\frac{m}{2}$ pairs and use for each pair the procedure presented above (Figure 11) [25]. If instead, $m$ is odd, then we can add an extra qubit in the circuit, so that we can group again all qubits into pairs and repeat the same algorithm.
The last remaining problem occurs in the case where in $Q$ we have 2 consecutive $\text{CCSIGN}$ gates applied on the same qubits (call them $x, y, z$). In this situation, we would need 2 functions equal to $(-1)^{x \cdot y \cdot z}$. To get rid of the Hadamards in between the $\text{CCSIGN}$ operators, we insert another function $f'$, such that $f_j = (-1)^{x \cdot y \cdot z}$, $f_{j+1} = f'$, $f_{j+2} = (-1)^{x \cdot y \cdot z}$ and $f' = 1$. Consequently, the Hadamards between $f_j$ and $f_{j+1}$ will be cancelled by the Hadamards between $f_{j+1}$ and $f_{j+2}$ and we will remain with 2 consecutive $\text{CCSIGN}$ gates [1, 29].

All in all, we simulated each $\text{CCSIGN}$ gate of the circuit $Q$ using 1 function in the $\text{Forrelation}$ problem and each $H$ gate using 3 functions.

Therefore, starting from a quantum circuit $Q$ containing $M$ gates: Hadamards or $\text{CCSIGN}$, we are able to find $k$ functions, $f_1, f_2, ..., f_k$ such that the $\text{Forrelation}$ circuit simulates the circuit of $Q$ ($S_Q = \Theta_{f_1, f_2, ..., f_k}$), where $k = O(M)$. Moreover, we obtained that any of the $k$ functions is of the form $f_i: \{0,1\}^{n+1} \rightarrow \{−1, 1\}$ $f_i(x_1, x_2, ..., x_{n+1}) = (-1)^{x_a \cdot x_b}$ or $f_i(x_1, x_2, ..., x_{n+1}) = (-1)^{x_u \cdot x_v \cdot x_w}$ (the input has length $n + 1$, because we might need to add an extra bit to solve the Hadamard issue described above).

Then, we have found a reduction in polynomial time from a $BQP$–complete problem (QSIM) to $k$–Forrelation, when $k = O(n)$. Additionally, we have given a $BQP$ algorithm to this problem, thus, we have shown that $k$–Forrelation is in $BQP$–complete. □

At this point, we could not find a way to relate the $k$–Forrelation to the class $NP/poly$. Instead, we are now going to link this $BQP$–complete problem to a similar oracle problem [25, 30] which is easier to work on.

8.3. Real Forrelation

**Definition 11.** In the Real Forrelation problem [25] we are given 2 functions $f, g: \{0, 1\}^n \rightarrow \mathbb{R}$ and we can obtain values of these functions in different points through an oracle $O$. Given the fact that there are possible 2 scenarios involving the 2 input functions, we are asked to determine in which of the 2 cases we are. The 2 scenarios are the next ones:

i. For all $x, y \in \{0, 1\}^n$, $f(x)$ and $g(y)$ are both drawn from normal distributions with mean 0 and variance 1

ii. For all $x \in \{0, 1\}^n$, $f(x)$ are drawn from the normal distribution, but function $g$ depends on the values of $f$: $g(x) = \frac{1}{\sqrt{2\pi}} \sum_y \epsilon_{\{0,1\}^n} f(y) \cdot (-1)^{x \cdot y}$

The second case can be rewritten as: $g = H \cdot f$, where we considered $f$ and $g$ as 2 vectors containing the values of the functions and $H$ is the Hadamard matrix [4].

What we want to show is the following essential property [25]:

**Theorem 3.2.1.** Real Forrelation is at most as difficult to solve as the Forrelation problem. Then, by showing that we can obtain a negative answer for Real Forrelation being solved by a $TM$ in $NP/poly$, this would imply the same negative result for Forrelation.

**Proof.** Showing that Forrelation is at least as difficult as Real Forrelation can be achieved by finding a polynomial reduction from Real Forrelation to Forrelation (Real Forrelation $\leq_{poly}$ Forrelation) [20].
Given an input for the Real Forrelation problem, consisting of 2 real functions, we need to create an instance of the Forrelation problem.

In other words, from $f, g : \{0, 1\}^n \rightarrow \mathbb{R}$ we need to create 2 Boolean functions $F, G : \{0, 1\}^n \rightarrow \{-1, 1\}$ such that if $f, g$ are uncorrelated then we have $|\Theta_{F,G}| \leq \frac{1}{100}$ and if $g = Hf$ we have $\Theta_{F,G} \geq \frac{3}{5}$.

We build $F$ and $G$ in the following way:

$$F(x) = \text{sign}(f(x)) \quad \forall x \in \{0, 1\}^n$$

$$G(x) = \text{sign}(g(x)) \quad \forall x \in \{0, 1\}^n$$

Now, given that $g = Hf$, we need to show that $|\Theta_{F,G}| \leq \frac{1}{100}$

$$\Theta_{F,G} = \frac{1}{2^n} \cdot \sum_{x,y \in \{0,1\}^n} (-1)^{xy} F(x)G(y)$$

$$g(x) = (Hf)_x = (-1)^{x_0} \cdot f(0) + (-1)^{x_1} \cdot f(1) + \cdots + (-1)^{x_{2^n-1}} \cdot f(2^n-1) = \sum_{i=0}^{2^n-1} (-1)^{x_i} \cdot f(i)$$

We compute the estimated value [34] of each term of $\Theta_{F,G}$:

$$T_{x,y} = E[(-1)^{xy} \cdot F(x) \cdot G(y)] = E[(-1)^{xy} \cdot \text{sign}(f(x)) \cdot \text{sign}(\sum_{i \in \{0,1\}^n} (-1)^{y_i} \cdot f(i))]$$

We can rewrite $T_{x,y}$ as $T_{x,y} = E[-(-1)^{xy} \cdot \text{sign}(f(x)) \cdot \text{sign}(C + (-1)^{y-x} \cdot f(x))]$, where

$$C = \sum_{i=0}^{2^n-1} (-1)^{y_i} \cdot f(i).$$

Because we know that all values of $f$ are independently drawn from $\mathcal{N}(0, 1)$, we have that $f(x)$ and $C$ are also independent [33, 34] (as $C$ involves every value of $f$ except the one on input $x$) and hence:

$$E[-(-1)^{xy} \cdot \text{sign}(f(x)) \cdot \text{sign}(C)] = 0$$

Now, when we add $(-1)^{y-x} \cdot f(x)$ to $C$ there are 2 possibilities:

a) $\text{sign}(C + (-1)^{y-x} \cdot f(x)) = \text{sign}(C)$, so $T_{x,y} = E[-(-1)^{xy} \cdot \text{sign}(f(x)) \cdot \text{sign}(C)] = 0$

b) $\text{sign}(C + (-1)^{y-x} \cdot f(x)) = \text{sign}((-1)^{y-x} \cdot f(x))$ and $\text{sign}((-1)^{y-x} \cdot f(x)) \neq \text{sign}(C)$

Therefore, we obtain that $T_{x,y} = 2 \cdot P(\text{sign}(C + (-1)^{y-x} \cdot f(x)) \neq \text{sign}(C))$

$\text{sign}(C + (-1)^{y-x} \cdot f(x)) \neq \text{sign}(C)$ can only happen when:

$\text{sign}(C) \neq \text{sign}((-1)^{y-x} \cdot f(x))$ and $|C| < |(-1)^{y-x} \cdot f(x)|$.

Consider $C(t)$ the continuous random variable of $C$ [34]. Then, we have that:

$$T_{x,y} = 2 \cdot \int_{-\infty}^{+\infty} P(|t| < |(-1)^{y-x} \cdot f(x)|) \cdot C(t) dt = 4 \cdot \int_{0}^{+\infty} P(t < (-1)^{y-x} \cdot f(x)) \cdot C(t) dt$$

Since $C = \sum_{i=0}^{2^n-1} (-1)^{y_i} \cdot f(i)$ and each $f(i)$ represents a Gaussian $\mathcal{N}(0, 1)$, we obtain that $C$ is also a Gaussian with mean 0 and variance $2^n - 1$, namely that $C(t) = \frac{1}{\sqrt{2\pi(2^n-1)}} \cdot e^{-\frac{t^2}{2(2^n-1)}}$.

Moreover, due to the fact that the mean of $f$ is 0 and that $(-1)^{y-x}$ can only take values $-1$ or 1 it suffices to consider the case $(-1)^{y-x} = 1$. 

63
Hence, \( T_{x,y} = 4 \cdot \int_0^{+\infty} P(t < f(x)) \cdot \frac{1}{\sqrt{2\pi(2^2-1)}} \cdot e^{-\frac{t^2}{2(2^2-1)}} dt \)

Using the fact that the function \( e^{-\frac{t^2}{2(2^2-1)}} \) is strictly decreasing on the interval \((0, \infty)\), then \( e^{-\frac{t^2}{2(2^2-1)}} < 1 \) and we get:

\[
T_{x,y} \leq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot \int_0^{+\infty} P(t < f(x)) dt
\]

But, \( \int_0^{+\infty} P(t < f(x)) dt = E(f(x)) \)

As any value \( f(x) \) is an independent \( N(0,1) \) its probability density function [33] is equal to \( \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \)

Thus, \( E(f(x)) = \int_0^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} \cdot t dt = \frac{1}{\sqrt{2\pi}} \cdot \left(-\frac{t^2}{2}\right) \bigg|_0^{\infty} = \frac{1}{\sqrt{2\pi}} \cdot \left(1 - e^{-\infty}\right) = \frac{1}{\sqrt{2\pi}} \)

Therefore, \( T_{x,y} \leq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot \frac{1}{\sqrt{2\pi}} = \frac{2}{\pi \sqrt{2n-1}} \leq 2 \pi \sqrt{2n-1} + O \left( \frac{1}{\sqrt{2n}} \right) \quad \forall x, y \in \{0,1\}^n \)

On the other hand, for any constant \( D > 0 \) we have that:

\[
T_{x,y} = \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot \left[ \int_0^D P(t < f(x)) \cdot e^{-\frac{t^2}{2(2^2-1)}} dt + \int_D^{+\infty} P(t < f(x)) \cdot e^{-\frac{t^2}{2(2^2-1)}} dt \right]
\]

Then, \( T_{x,y} \geq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot \int_0^D P(t < f(x)) \cdot e^{-\frac{t^2}{2(2^2-1)}} dt \geq 0 \)

Again, using that the function \( e^{-\frac{t^2}{2(2^2-1)}} \) is strictly decreasing on the interval \((0, \infty)\), we obtain that:

\[
T_{x,y} \geq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot e^{-\frac{D^2}{2(2^2-1)}} \cdot \int_0^D P(t < f(x)) dt = \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot e^{-\frac{D^2}{2(2^2-1)}} \cdot (\int_0^D P(t < f(x)) dt) - \int_D^{+\infty} P(t < f(x)) dt
\]

\[
T_{x,y} \geq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot e^{-\frac{D^2}{2(2^2-1)}} \cdot \left( \int_0^{+\infty} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2}} \cdot t dt - \int_D^{+\infty} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2}} \cdot t dt \right)
\]

\[
T_{x,y} \geq \frac{4}{\sqrt{2\pi(2^2-1)}} \cdot e^{-\frac{D^2}{2(2^2-1)}} \cdot \left( \frac{1}{\sqrt{2\pi}} - \frac{e^{-\frac{D^2}{2}}}{\sqrt{2\pi}} \right) = \frac{2}{\pi \cdot \sqrt{2n}} \cdot e^{-\frac{D^2}{2(2^2-1)}} \cdot \left(1 - e^{-\frac{D^2}{2}}\right)
\]

If we choose \( D = \sqrt{2n} \) we will obtain:

\[
T_{x,y} \geq \frac{2}{\pi \sqrt{2n}} \cdot e^{-\frac{n}{2(2^2-1)}} \cdot \left(1 - e^{-n}\right) \geq \frac{2}{\pi \sqrt{2n}} - O \left( \frac{n}{\sqrt{2n}} \right) \quad \forall x, y \in \{0,1\}^n
\]

\( \Theta_{F,G} \) has \( 2^n \cdot 2^n \) terms \( T_{x,y} \), therefore we reach that:

\[
E(\Theta_{F,G}) \leq \frac{1}{2^n} \cdot 2^n \cdot \left( \frac{2}{\pi \sqrt{2n}} + O \left( \frac{1}{\sqrt{2n}} \right) \right) = \frac{2}{\pi} + O \left( \frac{1}{2^n} \right)
\]
\[ E(\theta_{F,G}) \geq \frac{1}{\pi} \cdot 2^n \cdot 2^n \cdot \left( \frac{2}{\pi \sqrt{2\pi}} - O \left( \frac{n}{\pi \sqrt{2\pi}} \right) \right) = \frac{2}{\pi} - O \left( \frac{n}{\pi \sqrt{2\pi}} \right) \tag{2} \]

From (1) and (2) we get \( E(\theta_{F,G}) \approx \frac{2}{\pi} \geq \frac{3}{5} [32] \), which is what we wanted to show.

The second part of the reduction consists of proving that if every \( f(x) \) and \( g(y) \) are uncorrelated, then \( |\theta_{F,G}| \leq \frac{1}{100} \).

In this case, \( T_{x,y} = E[(-1)^{x \cdot F(x) \cdot G(y)}] = 0 \) \( \forall x, y \in \{0,1\}^n, x \neq y \) (as now \( F(x) \) and \( G(y) \) are also independent [34]).

Then, in the \( \theta_{F,G} \) sum we are left with only \( 2^n \) terms, so we obtain:

\[ E[\theta_{F,G}] = \frac{1}{\pi} \cdot 2^n = \frac{\pi}{2^n} < \frac{1}{100}. \]

Consequently, we have shown that there exists a reduction from the Real Forrelation problem to Forrelation. \( \square \)

Because we need to relate Real Forrelation to the \textit{k-Forrelation} problem (only the \textit{k-Forrelation} problem is \textit{BQP-complete} when \( k = O(n) \)), we also build a version of the Real Forrelation problem involving \( k \) functions.

**Definition 12.** In the \textit{k-Real Forrelation} [25] we have a sequence of \( k \) functions \( f_1, f_2, \ldots, f_k \), where \( f_1, f_2, \ldots, f_{k-2} : \{0,1\}^n \to \{-1,1\} \) and \( f_{k-1}, f_k : \{0,1\}^n \to \mathbb{R} \). We know that all values \( f_i(x) \) \( \forall i \in \{1, \ldots, k-2\} \) are independently drawn from the set \( \{-1,1\} \). Moreover, we are promised that for \( f_{k-1}, f_k \), one of the following 2 cases hold:

i) Every \( f_{k-1}(x) \) and \( f_k(y) \) are both drawn from 2 normal distributions \( \forall x, y \in \{0,1\}^n \)

ii) Every \( f_{k-1}(x) \) is drawn from a normal distribution, but the function \( f_k \) depends on the values of all other \( k - 1 \) functions:

\[ f_k(x) = \frac{1}{2^{(k-1)n}} \sum_{x_1, \ldots, x_{k-1} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2 \cdot \ldots \cdot x_{k-1}} \cdot f_1(x_1) \cdot \ldots \cdot f_{k-1}(x_{k-1}) \]

As before, the problem asks to determine which of the cases applies given a sequence of \( k \) functions.

**Theorem 3.2.2.** \textit{k-Real Forrelation} is equivalent to the problem Real Forrelation [25].

**Proof.** Consider \((f_1, f_2, \ldots, f_k)\) an instance of the \textit{k-Real Forrelation}. Then, we can build 2 functions \( f, g : \{0,1\}^n \to \{-1,1\} \) in the following way:

\[ g(x) = f_k(x) \]

\[ f(x) = f_{k-1}(x) \cdot d_x, \text{ with} \]

\[ d_x = \frac{1}{2^{(k-2)n}} \sum_{x_1, \ldots, x_{k-2} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2 \cdot \ldots \cdot x_{k-2}} \cdot f_1(x_1) \cdot \ldots \cdot f_{k-2}(x_{k-2}) \cdot (-1)^{x_{k-2} \cdot x} \]
We make the first observation that: \( \theta_{f,g} = \theta_{f_1,f_2,...,f_k} \). Indeed, if we replace \( f \) and \( g \) with their definitions in \( \theta_{f,g} \), we get:

\[
\theta_{f,g} = \frac{1}{2^n} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} g(y) \cdot f(x) = \frac{1}{2^n} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} f_k(y) \cdot f_{k-1}(x) \cdot d_x =
\]

\[
= \frac{1}{2^n} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} f_k(y) \cdot f_{k-1}(x) \cdot \left[ \frac{1}{2^{(k-1)n}-1} \sum_{x_1,...,x_{k-2} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdots (-1)^{x_{k-3} \cdot x_{k-2}} \right] f_1(x_1) \cdots f_{k-2}(x_{k-2}) = (-1)^{x_{k-2} \cdot x}
\]

\[
= \frac{1}{2^{(k-1)n}-2} \sum_{x_1,...,x_{k-2} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdots (-1)^{x_{k-2} \cdot x} f_k(y) \cdot f_{k-1}(x) \cdot f_1(x_1) \cdots f_{k-2}(x_{k-2}) = \theta_{f_1,f_2,...,f_k}
\]

Now if \( f_{k-1}(x) \) and \( f_k(x) \) are both uncorrelated normal distributions, then because \( g = f_k \) and \( f \) is a linear combination of values of the functions \( f_1, f_2, ..., f_{k-1} \), we obtain that every \( f(x) \) and \( g(x) \) values are uncorrelated normal distributions \([28, 34]\).

What is left to show is that if the functions \( \{f_1(x), ..., f_{k-1}(x), f_k(x)\} \) are in the second case of \( k\)-\textit{Real Forrelation}, then \( f \) and \( g \) are also in the second scenario of \textit{Real Forrelation}.

We have \( g(x) = f_k(x) = \frac{1}{2^{(k-1)n}-2} \sum_{x_1,...,x_{k-1} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdots (-1)^{x_{k-2} \cdot x} f_1(x_1) \cdots f_{k-1}(x_{k-1}) \)

\[
g(x) = \sum_{x \in \{0,1\}^n} f_k(x) \cdot \frac{1}{2^{(k-1)n}-2} \sum_{x_1,...,x_{k-2} \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdots (-1)^{x_{k-2} \cdot x} f_1(x_1) \cdots f_{k-2}(x_{k-2}) = (-1)^{x_{k-2} \cdot x}
\]

\[
g(x) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} f_k(x) \cdot (-1)^{x_{k-1} \cdot x}
\]

Therefore, we have proved that the \( k\)-\textit{Real Forrelation} problem is equivalent to the problem \textit{Real Forrelation} (with only 2 functions).

Now, we can finally show that \( k\)-\textit{Real Forrelation} can be reduced to \( k\)-\textit{Forrelation}.

\textbf{Proof.} Given an instance of \( k\)-\textit{Real Forrelation}, \( f_1, f_2, ..., f_{k-2} : \{0,1\}^n \rightarrow \{-1,1\} \) and \( f_{k-1}, f_k : \{0,1\}^n \rightarrow \mathbb{R} \), we build an instance of \( k\)-\textit{Forrelation} \( F_1, F_2, ..., F_k : \{0,1\}^n \rightarrow \{-1,1\} \) in this way:

\[F_1(x) = f_1(x), \ldots, F_{k-2}(x) = f_{k-2}(x)\]

\[F_{k-1}(x) = \text{sgn}(f_{k-1}(x)), F_k(x) = \text{sgn}(f_k(x))\]

Then, \( \theta_{F_1,F_2,...,F_k} = \frac{1}{2^{(k+1)n}-2} \sum_{x_1,x_2,...,x_k \in \{0,1\}^n} (-1)^{x_1 \cdot x_2} \cdots (-1)^{x_{k-1} \cdot x_k} \cdot f_1(x_1) \cdots f_{k-2}(x_{k-2}) \cdot f_{k-1}(x_{k-1}) \cdot f_k(x_k) \cdot d_{x_{k-1}}\)

Next, we can take the expected value of each term of this sum, \( E(F_{k-1}(x) \cdot F_k(y) \cdot d_x) \) and show as we did for the \textit{Real Forrelation} – \textit{Forrelation} reduction that \( E(\theta_{F_1,F_2,...,F_k}) \equiv \frac{2}{n} \geq \frac{3}{5} \) when \( f_k \) depends on the values of \( f_1, f_2, \ldots, f_{k-1} \) and \( E(\theta_{F_1,F_2,...,F_k}) \equiv 0 \leq \frac{1}{100} \) when \( f_k \) values are uncorrelated.

Therefore, we have obtained that \( k\)-\textit{Forrelation} is harder to solve than \( k\)-\textit{Real Forrelation}, which in turn is equivalent to \textit{Real Forrelation}. 

\[\square\]
At this point, the remaining part is to show that \textit{Real Forrelation} cannot be solved by any machine in \( NP/poly^0 \).

8.3.1. Distribution Correlation

Instead of working with \textit{Real Forrelation}, we consider an abstraction of this problem, called \textit{Distribution Correlation} [25].

\textbf{DEFINITION 13.} In \textit{Distribution Correlation} we have a set of \( 2^{n+1} \) vectors, \( \mathcal{W} = \{ w_1, w_2, ..., w_{2^{n+1}} \} \), where \( w_i \in \mathbb{R}^2 \). This problem is a “black box” model [23], for each vector \( w_i \) queried we receive in response a random number \( c_i \), with the promise that either:

\begin{itemize}
  \item[i)] all responses for all vectors \( \{ c_1, c_2, ..., c_{2^{n+1}} \} \) are independently chosen from a normal distribution or
  \item[ii)] For every response, we have that: \( c_i = (w_i, v) \) - the inner vector between \( w_i \) and \( v \), where \( v \) is a fixed vector from \( \mathbb{R}^{2^{n+1}} \).
\end{itemize}

The problem asks to determine if \( \{ c_1, c_2, ..., c_{2n} \} \) are chosen from the normal distribution (case i) or if there exists a \textit{positive covariance} [28] between them (case ii).

The complexity reduces to the number of queries we have to make in order to decide in which of the 2 situations we are.

\textbf{OBS 10.} We notice that we can solve this problem deterministically using at most \( 2^n + 1 \) queries. We can view \textit{Distribution Correlation} as the problem of deciding if there exists a vector \( v \in \mathbb{R}^2 \), such that for any response \( c_i \), we have that \( c_i = (w_i, v) \). Suppose there exists such a vector \( v \).

Then, for each query we make, we get a new linear equation involving the vector \( v \):

\[
\begin{align*}
(w_1, v) &= c_1 \\
(w_2, v) &= c_2 \\
&\vdots \\
(w_{2^n}, v) &= c_{2^n}
\end{align*}
\]

Consequently, after \( 2^n \) queries we obtain a system of \( 2^n \) linear equations containing the \( 2^n \) elements of \( v \). Therefore, we can solve the resulting linear system and obtain \( v \). To determine in which of the 2 cases of the problem we are, all we need to do is to make one more query for the vector \( w_{2^{n+1}} \). Then, we check if the response we receive from the oracle, \( c_{2^{n+1}} \), verifies the relation: \( c_{2^{n+1}} = (w_{2^{n+1}}, v) \) (with the \( v \) computed before). If it does, this implies that there actually exists a positive covariance between the oracle’s responses, so we must be in the second case. Otherwise, it means that \textit{not} all query responses \( c_i \) verify the property that there exists a vector \( v \) such that: \( c_i = (w_i, v) \), so we are in the former situation.

\textbf{OBS 10.} Analysing the \textit{Distribution Correlation} problem, we deduce it is similar to \textit{Simon’s} problem. \textit{Simon’s} problem can be solved if we show the existence of the \textit{bit-mask} string \( s \), while the \textit{Distribution Correlation} problem can be decided if we prove the existence of the vector \( v \).
Furthermore, in Simon’s problem we queried the oracle for the input $x_i$ to obtain the value of the function in that point, $f(x_i)$. Equivalently in the current problem we query $w_i$ to obtain the corresponding random value $c_i$.

Both Simon and Distribution Correlation can only be solved by Oracle Turing Machines [10].

Thus, any machine $M$ solving the Distribution Correlation problem would have in its computational tree at each node a query $w_i$ to the oracle. Then, based on all the query answers $(c_1, ..., c_i)$ along a path in its computational tree, $M$ will decide whether all $c_i$’s are independent or not.

Considering all these arguments, we can follow the same proofs we used for **Main Theorem 2.2** and **Main Theorem 2.3**, in order to show the two results below.

**Corollary 2:** Distribution Correlation $ \in \mathsf{NP}^0$.

**Proof Sketch.** The oracle receives the dimensions of our vectors, $2^n$, and the vector with the associated index $i$, and returns $c_i$. We can show that Distribution Correlation cannot be solved by a nondeterministic machine using diagonalization [21]. For each nondeterministic TM $M_i$, the oracle $O$ finds a vector of responses $(c_1, ..., c_{2^n})$ which is accepted by $M_i$. Because the adversarial oracle knows the computational tree of $M_i$, he can trick $M_i$ in giving the wrong decision [31]. $M_i$ accepts an input $I$ is equivalent to saying that there exists a path $\pi$ in his computational tree containing a polynomial number of queries $\{w_{j_1}, w_{j_2}, ..., w_{j_k}\}$ (with their associated answers $(c_{j_1}, c_{j_2}, ..., c_{j_k})$), such that $\pi$ accept $I$. Then, the oracle $O$ can keep the answers to these $k$ queries, but can still set the values for the rest of the answers $c_i$ such that there exists a vector $v$ with the property that $\langle w_{j_i}, v \rangle = c_{j_i} \forall i \in \{1, ..., 2^n\}$. By fixing the answers $(c_{j_1}, c_{j_2}, ..., c_{j_k})$ we just set $k$ out of $2^n$ elements of $v$, so no matter what $M_i$ queries on path $\pi$, the oracle can deceive him to accept the input.

**Corollary 3:** Distribution Correlation $ \in \mathsf{NP}/\mathsf{poly}^0$.

**Proof Sketch.** For the $\mathsf{NP}/\mathsf{poly}$ case we take into consideration every polynomial-size advice a nondeterministic machine $M_i$ can receive [32, 33]. For each of them we basically have a new nondeterministic machine which could solve Distribution Correlation. Our aim is to show that we can eliminate all these machines by properly setting a different set of answers $(c_1, ..., c_{2^n})$ for each input index. Because $M_i$ receives the same advice for all sets of vectors from $\mathbb{R}^{2^n}$, for every input to $M_i$, we can eliminate a fraction of the total number of these advices the same way we did for the coSimon problem [23].

In this section we have presented another candidate problem which suggests the oracle separation between $\mathsf{BQP}$ and $\mathsf{NP}/\mathsf{poly}$. We have started from a $\mathsf{BQP} − \mathsf{complete}$ problem, $k$-Forrelation, and through a sequence of equivalences and reductions between problems, we have reached the Distribution Correlation problem. By observing a series of similarities between Distribution Correlation and Simon’s problem, we applied similar arguments to indicate that Distribution Correlation cannot belong to the class $\mathsf{NP}/\mathsf{poly}^0$.

Because the chain of relations described in **Theorem 3.2.1** and **Theorem 3.2.2** tell us that Distribution Correlation is at most as “difficult” as the $\mathsf{BQP} − \mathsf{complete} k$-Forrelation problem, this would imply that the no-go result also holds for $k$-Forrelation.

Consequently, because $\mathsf{BQP} − \mathsf{hard}$ represents the most difficult problems from $\mathsf{BQP}$ [19], we infer that for any problem $\mathcal{P}$ in $\mathsf{BQP} − \mathsf{hard}$ there is no oracle $\mathsf{NP}/\mathsf{poly}$ machine that can solve $\mathcal{P}$.
Quantum technologies are evolving at a rapid pace motivated by the staggering potential of quantum computers to reach a computational efficiency unattainable by any classical machine [3].

The prospect of such a powerful computer puts us in the position to address the following vital problem: To what extent can we take advantage of the computational resources of a quantum computer in a way that our data is protected?

In this paper we have analysed the encryption scheme GES which allows a classical client to delegate an unfeasible computation to a quantum server and derive the result of the computation such that his input remains private from the server.

We illustrated the usefulness of the GES framework by defining encryption schemes which solve efficiently some hard problems (discrete logarithm, primitive root, quadratic residuosity) [16].

Then, we focused on the main theme of the project: the relation between the difficulty of encrypting a function using a GES protocol and the difficulty of solving it. We studied this problem from a complexity theoretic point of view. Firstly, we reviewed the recent work of Abadi who proved that there is a connection between encryptability and nonuniform complexity [16, 24].

Based on this result, we investigated if the class of problems efficiently solvable by a quantum computer (BQP problems) admits a GES protocol. We conjecture that not all BQP problems can be solved in the GES scenario. This would be equivalent to showing an outstanding complexity theory result: BQP is not included in the nonuniform class NP/poly. However, proving that BQP is not contained in NP/poly is at least as difficult as showing that P is not contained in NP [17, 20]. Consequently, we decided to give strong arguments in favour of the separation between the 2 classes by showing that BQP cannot be included in NP/poly with respect to an oracle (we relativized [31] the relation between the quantum and the classical complexity class).

To prove the oracle separation between BQP and NP/poly we considered two candidates: Simon’s problem and Forrelation. We first proved that the two problems can be solved using BQP algorithms. Then, using an innovative method akin to an exhaustive search, we showed that we can define an oracle O such that no possible polynomial advice could help an NP machine solve these problems. Consequently, they cannot belong to the class NP/polyO. Finally, because all problems from BQP – Hard are more difficult than any problem from BQP (including Simon and Forrelation), this implies that none of these problems can be solved by an NP/polyO algorithm.

As a result, we presented powerful evidence that BQP – Hard problems cannot be solved in this secure delegated computation task.

Our second contribution was related to the UBQC quantum protocol [26] which allows a client possessing some minimal quantum abilities, to determine the result of a hard computation by communicating with a quantum computer, while keeping the input and the computation itself private.

Inspired by the idea of having a purely classical client who could benefit from the security advantages guaranteed by UBQC, we investigated the existence of a classical version of UBQC (CUBQC). We proved that this resulting scheme could be represented as an instance of the GES. Therefore, CUBQC inherits all the complexity theoretic results known for GES.
All these results achieved in our project inspire us to pose the following open problems:

- Do the no-go theorems regarding the encryptability of \( NP \) – \textit{Hard} and \( BQP \) – \textit{Hard} functions still hold if we modify the security conditions of the \textit{GES} framework (for instance allowing the server to infer more than just the size of the input)?

- Consider a \textit{quantum generalization} of the \textit{GES} scheme, \textit{QGES}, where the client has some limited quantum abilities, but still requires the superior computational abilities of a quantum server. Can we derive quantum analogue complexity theory results for \textit{QGES}; namely, can we derive that in the \textit{QGES} scenario any \( BQP \) function can be encrypted without leaking anything and that any encryptable function can be solved by a \( QNP/poly \) algorithm?

In conclusion, we accomplished the goals we set for this project and the no-go results we indicated lead to many new questions which must be dealt with, in order to fully understand what are the limitations imposed by a secure exploit of the high performance of a quantum computer.
Bibliography


