



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

# **SearchQS: Una Piattaforma Web per l'Individuazione ed Esplorazione di Problemi di Progettazione in Circuiti Quantistici.**

RELATORE

Prof. Fabio Palomba

Università degli Studi di Salerno

CANDIDATO

**Gianluca Scisciolo**

Matricola: 0512107432

Anno Accademico 2023-2024

*Questa tesi è stata realizzata nel*

sesa<sup>lab</sup>  
SOFTWARE ENGINEERING  
SALERNO

*"Nature is quantum, goddamn it! So if we want to simulate it, we need a quantum computer."*

*"La natura è quantistica, maledizione! Quindi, se vogliamo simularla, abbiamo bisogno di un computer quantistico."*

*Richard Feynman - maggio 1981 - Prima Conferenza sulla Fisica della Computazione*

## Abstract

L'**informatica quantistica** è una scienza che si basa sulla matematica classica, meccanica quantistica e logica quantistica, a differenza dell'**informatica classica** che si basa sulla matematica classica, fisica classica e logica classica.

I primi studi sul calcolo quantistico sono stati fatti nel 1935 grazie ad un progetto di collaborazione a cui partecipò anche Albert Einstein.

L'idea di un computer quantistico, però, nacque nel 1980, quando il fisico Paul Benioff e il matematico Yuri Manin ne parlano in alcune pubblicazioni indipendenti.

L'informatica quantistica poi prese piede nella comunità scientifica attorno agli anni '90 del 1900 per poi iniziare ad essere noto anche al di fuori della comunità scientifica con la creazione dei primi computer quantistici attorno al primo decennio degli anni 2000.

L'informatica quantistica, è una scienza studiata principalmente da matematici, fisici ed informatici e, poiché i primi 2 non hanno delle solide basi in ingegneria del software, tendono a creare del software quantistico senza tenere in considerazione l'ingegneria del software studiata principalmente dagli informatici, questo comporta la risoluzione di alcuni problemi che vengono trattati anche dall'informatica classica, uno di questi problemi sono i così detti Quantum Code Smells che, nell'informatica classica, si chiamano Classical Code Smells.

Per questo motivo, durante il lavoro svolto con il torocinio interno, ho progettato ed implementato una Web Application intitolata SearchQS che ha l'obiettivo di analizzare un sistema quantistico inserito in input da un utente per individuare dei possibili Quantum Code Smells presenti nel sistema caricato.

In questa tesi verranno riportate alcuni concetti di matematica e meccanica quantistica utili per capire l'informatica quantistica e verrà riportata una descrizione dell'informatica quantistica, dei Quantum Code Smells che verranno individuati e per finire verranno riportati i quantum code smells individuati in 3 sistemi quantistici che ho caricato in input contenenti alcuni algoritmi quantistici noti.

---

## Indice

---

<b>Elenco delle Figure</b>	<b>iii</b>
<b>Elenco delle Tabelle</b>	<b>vi</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Contesto applicativo . . . . .	1
1.2 Motivazioni e obiettivi . . . . .	2
1.3 Risultati ottenuti . . . . .	3
1.4 Struttura della tesi . . . . .	4
<b>2 Background e Stato dell'arte</b>	<b>6</b>
2.1 Informatica . . . . .	6
2.2 Concetti di matematica . . . . .	8
2.3 Concetti di meccanica quantistica . . . . .	12
2.3.1 Sovrapposizione quantistica . . . . .	12
2.3.2 Entanglement ed entangled . . . . .	14
2.3.3 Interferenza quantistica . . . . .	15
2.4 Potenza del calcolo quantistico e numero di stati possibili . . . . .	16
2.5 Transpilazione quantistica . . . . .	17
2.6 Porte logiche quantistiche . . . . .	19
2.7 Code Smells . . . . .	20
2.8 QSmell e SearchQS . . . . .	25

<b>3</b>	<b>SearchQS: Una Piattaforma Web per l'Individuazione ed Esplorazione di Problemi di Progettazione in Circuiti Quantistici.</b>	<b>27</b>
<b>4</b>	<b>Analisi di alcuni sistemi quantistici</b>	<b>34</b>
4.1	Sistema VariAlgoritmi . . . . .	34
4.2	Sistema AlgoritmiAI . . . . .	44
4.3	Sistema AlgoritmiSicurezza . . . . .	51
4.4	Risultati quantum code smells ottenuti . . . . .	56
<b>5</b>	<b>Conclusioni</b>	<b>63</b>
5.1	Sviluppi futuri . . . . .	63
5.2	Conclusioni . . . . .	65
	<b>Bibliografia</b>	<b>67</b>

---

## Elenco delle figure

---

2.1	Transistor . . . . .	12
2.2	1 atomo di silicio con 14 elettroni . . . . .	13
2.3	Esempio di spin-up e spin-down . . . . .	14
2.4	Bit e Qubit . . . . .	14
2.5	Esempio di interferenza quantistica . . . . .	16
2.6	Gate X . . . . .	19
2.7	Gate Z . . . . .	19
2.8	Gate CX . . . . .	19
2.9	Gate CZ . . . . .	20
2.10	Gate H . . . . .	20
2.11	Esempio di circuito logico quantistico . . . . .	20
4.1	Circuito teleportation_circuit senza transpilazione . . . . .	35
4.2	Matrice teleportation_circuit senza transpilazione . . . . .	36
4.3	Circuito teleportation_circuit original . . . . .	36
4.4	Matrice teleportation_circuit original . . . . .	36
4.5	Circuito teleportation_circuit rpcx . . . . .	36
4.6	Matrice teleportation_circuit rpcx . . . . .	37
4.7	Circuito teleportation_circuit simple . . . . .	37
4.8	Matrice teleportation_circuit simple . . . . .	37
4.9	Circuito teleportation_circuit ibm_perth . . . . .	37
4.10	Matrice teleportation_circuit ibm_perth . . . . .	37

4.11 Circuito teleportation_circuit ibm_sherbroke . . . . .	38
4.12 Matrice teleportation_circuit ibm_sherbroke . . . . .	38
4.13 Circuito random_binary_number_circuit senza transpilazione . . . . .	39
4.14 Matrice random_binary_number_circuit senza transpilazione . . . . .	39
4.15 Circuito random_binary_number_circuit original . . . . .	40
4.16 Matrice random_binary_number_circuit original . . . . .	40
4.17 Circuito random_binary_number_circuit rpcx . . . . .	41
4.18 Matrice random_binary_number_circuit rpcx . . . . .	41
4.19 Circuito random_binary_number_circuit simple . . . . .	42
4.20 Matrice random_binary_number_circuit simple . . . . .	42
4.21 Circuito random_binary_number_circuit ibm_perth . . . . .	43
4.22 Matrice random_binary_number_circuit ibm_perth . . . . .	43
4.23 Circuito random_binary_number_circuit ibm_sherbroke . . . . .	44
4.24 Matrice random_binary_number_circuit ibm_sherbroke . . . . .	44
4.25 Circuito quantum_point_classification_circuit senza transpilazione . . . . .	46
4.26 Matrice quantum_point_classification_circuit senza transpilazione . . . . .	46
4.27 Circuito quantum_point_classification_circuit original . . . . .	46
4.28 Matrice quantum_point_classification_circuit original . . . . .	47
4.29 Circuito quantum_point_classification_circuit rpcx . . . . .	47
4.30 Matrice quantum_point_classification_circuit rpcx . . . . .	47
4.31 Circuito quantum_point_classification_circuit simple . . . . .	48
4.32 Matrice quantum_point_classification_circuit simple . . . . .	48
4.33 Circuito quantum_point_classification_circuit ibm_perth . . . . .	49
4.34 Matrice quantum_point_classification_circuit ibm_perth . . . . .	49
4.35 Circuito quantum_point_classification_circuit ibm_sherbroke parte 1 . . . . .	50
4.36 Circuito quantum_point_classification_circuit ibm_sherbroke parte 2 . . . . .	50
4.37 Matrice quantum_point_classification_circuit ibm_sherbroke . . . . .	51
4.38 Circuito a_mod_15_circuit . . . . .	53
4.39 Matrice a_mod_15_circuit . . . . .	53
4.40 Circuito qft_inverse_circuit parte 1 . . . . .	53
4.41 Circuito qft_inverse_circuit parte 2 . . . . .	53
4.42 Circuito qft_inverse_circuit parte 3 . . . . .	54
4.43 Matrice qft_inverse_circuit . . . . .	54
4.44 Circuito shor_15_circuit senza transpilazione . . . . .	54



4.45 Matrice shor_15_circuit senza transpilazione . . . . .	55
---	----

---

## Elenco delle tabelle

---

2.1	Transpilazioni considerate . . . . .	18
2.2	Circuiti quantistici considerati . . . . .	25
4.1	Analisi statica nessuna transpilazione . . . . .	57
4.2	Analisi dinamica nessuna transpilazione . . . . .	57
4.3	Analisi statica transpilazione original . . . . .	58
4.4	Analisi dinamica transpilazione original . . . . .	58
4.5	Analisi statica transpilazione rpcx . . . . .	59
4.6	Analisi dinamica transpilazione rpcx . . . . .	59
4.7	Analisi statica transpilazione simple . . . . .	60
4.8	Analisi dinamica transpilazione simple . . . . .	60
4.9	Analisi statica transpilazione ibm_perth . . . . .	61
4.10	Analisi dinamica transpilazione ibm_perth . . . . .	61
4.11	Analisi statica transpilazione ibm_sherbroke . . . . .	62
4.12	Analisi dinamica transpilazione ibm_sherbroke . . . . .	62

# CAPITOLO 1

---

## Introduzione

---

### 1.1 Contesto applicativo

L'informatica quantistica è un campo innovativo che sfrutta i principi della meccanica quantistica per elaborare informazioni in modo radicalmente diverso rispetto ai computer tradizionali: mentre i computer classici utilizzano i bit che possono essere 0 o 1, i computer quantistici utilizzano i qubit, che possono essere 0, 1 o entrambi contemporaneamente grazie al fenomeno della sovrapposizione quantistica. Questa capacità permette ai computer quantistici di eseguire calcoli complessi molto più rapidamente. L'informatica quantistica è stata introdotta per superare i limiti dei computer classici. I computer tradizionali si basano su transistor che operano con i bit. Tuttavia, c'è un limite alla miniaturizzazione dei transistor: quando diventano troppo piccoli, entrano in gioco effetti quantistici che rendono il loro comportamento imprevedibile e inefficiente. Per risolvere quindi il problema della miniaturizzazione è stato introdotto il computer quantistico.

In sintesi, l'informatica quantistica è stata sviluppata per affrontare problemi complessi e per superare i limiti fisici della miniaturizzazione dei transistor nei computer classici.

Richard Feynman, durante la Prima Conferenza sulla Fisica della Computazione nel maggio 1981, ha pronunciato la seguente frase: "La natura è quantistica, maledizione!"

Quindi, se vogliamo simularla, abbiamo bisogno di un computer quantistico." per sottolineare un punto cruciale: la natura stessa opera secondo le leggi della meccanica quantistica. I fenomeni naturali, a livello microscopico, non seguono le leggi della fisica classica, ma quelle della fisica quantistica, intendeva che, per simulare accuratamente i processi naturali, dobbiamo utilizzare un computer che operi secondo i principi della meccanica quantistica, ovvero un computer quantistico. I computer classici, basati sulla fisica classica, non sono sufficientemente potenti per simulare i complessi fenomeni quantistici che avvengono in natura. In altre parole, Feynman stava sostenendo che per comprendere e simulare la natura in modo realistico, è necessario utilizzare strumenti che riflettano la sua vera natura quantistica.

Oggigiorno non abbiamo ancora dei computer quantistici abbastanza potenti da poter beneficiare in maniera totale dei grandi vantaggi della meccanica quantistica quindi, i ricercatori non possono fare un granché attualmente ma, in futuro, l'informatica quantistica verrà utilizzata in maniera migliore rispetto all'informatica classica in vari campi scientifici, tra cui:

- Machine Learning e Intelligenza Artificiale.
- Farmacologia.
- Chimica computazionale.
- Previsioni del tempo.
- Sicurezza informatica.

## 1.2 Motivazioni e obiettivi

Il progetto di tesi nasce dalla curiosità di capire come funziona l'informatica quantistica e perché è così diversa dall'informatica classica attualmente più conosciuta. Inoltre un altro motivo che mi ha spinto ad approfondire questo nuovo campo scientifico innovativo è il desiderio di cercare di ottenere una panoramica completa dell'informatica in generale. Siccome l'informatica quantistica è un campo molto complesso allora ho cercato di comprendere almeno le basi come:

- Introduzione all'informatica quantistica.

- Alcuni concetti della matematica che stanno alla base dell'informatica quantistica.
- Alcuni concetti della meccanica quantistica che stanno alla base dell'informatica quantistica.
- Potenza del calcolo quantistico e numero di stati possibili

Inoltre, il progetto di tesi si occuperà anche di provare la web application chiamata SearchQS progettata ed implementata durante il tirocinio interno svolto con il laboratorio SeSaLab situato all'Università degli studi di Salerno che consiste nel prendere in input un programma o sistema quantistico ed analizzarlo individuando i quantum code smells presenti nei vari file e nei vari circuiti quantistici. Il quantum code smell è un problema di progettazione in circuiti quantistici. Per fare ciò ho dovuto capire alcuni concetti tra cui:

- La transpilazione quantistica.
- Le porte logiche quantistiche.
- I Quantum Code Smells.

I vari concetti introdotti sopra verranno ripresi e spiegati nel capitolo 2.

## 1.3 Risultati ottenuti

Con la seguente tesi, ho approfondito le conoscenze relative all'informatica in generale introducendo le nuove conoscenze di base dell'informatica quantistica, in particolare, sono stati introdotti alcuni algoritmi:

- Algoritmi presenti nel sistema quantistico VariAlgoritmi:
  - Algoritmo per generare un numero pseudo-casuale (classico) e un numero casuale (quantistico).
  - Algoritmo per clonare una frase (classico) e algoritmo per teletrasportare un qubit oppure una frase (quantistico).
- Algoritmo nel campo dell'intelligenza artificiale presenti nel sistema AlgoritmiAI:

- Algoritmo di clusterizzazione k-means versione classica e versione ibrida.
- Algoritmi nel campo della sicurezza informatica presenti nel sistema Algoritmi-Sicurezza:
  - Algoritmo di fattorizzazione versione classica e versione quantistica (Shor15).
  - Algoritmo di crittografia classica (RSA) e attacco ad essa tramite la fattorizzazione classica di un numero.

Inoltre, con il lavoro di tirocinio interno, ho implementato per la prima volta un modo per poter analizzare un sistema scritto in Python che utilizza del codice quantistico scritto tramite Qiskit.

Inoltre, ho appreso nuove tecnologie, tra cui: Python, Qiskit e Flask.

Per quanto riguarda invece i risultati ottenuti con le analisi dei sistemi quantistici caricati in input, essi verranno riportati nella sezione 4.4.

## 1.4 Struttura della tesi

La tesi è articolata in cinque capitoli.

- **Capitolo 1:** In questo capitolo è presente l'introduzione relativa al lavoro di tesi sviluppato.
- **Capitolo 2:** In questo capitolo analizzeremo i concetti base dell'informatica quantistica e parleremo brevemente della differenza tra SearchQS e QSmell (sistema già presente prima della creazione di SearchQS).
- **Capitolo 3:** In questo capitolo risponderemo a delle domande di ricerca che ci siamo posti e, successivamente, andremo ad analizzare la web application SearchQS concentrandoci solamente sulla parte e sul codice che si occupa di analizzare un'applicazione o sistema quantistico inserito in input e sul codice che ci permetterà di individuare gli 8 quantum code smells che vogliamo analizzare.
- **Capitolo 4:** In questo capitolo andremo a provare la web application SearchQS analizzando i file contenenti gli algoritmi quantistici presenti nei sistemi quantistici analizzati. Inoltre, verranno riportati i circuiti quantistici analizzati con le

loro matrici e i risultati ottenuti sui file Python e sui circuiti quantistici dei file Python analizzati.

- **Capitolo 5:** Capitolo conclusivo che parla dei possibili sviluppi futuri della web application SearchQS e, verrà riportato cosa è stato fatto durante la tesi.

## CAPITOLO 2

---

### Background e Stato dell'arte

---

In questo capitolo daremo un breve accenno dei seguenti argomenti:

- Informatica quantistica.
- Alcuni concetti di matematica per l'informatica quantistica.
- Alcuni concetti di meccanica quantistica.
- Potenza del calcolo quantistico e numero di stati possibili.
- Transpilazione quantistica.
- Porte logiche quantistiche.
- Quantum Code Smells.

e parleremo della differenza tra i sistemi QSmell e SearchQS.

### 2.1 Informatica

L'**informatica (computer science)** possiamo vederla come l'unione dei termini informazione e automatica, ed è la scienza che si occupa del trattamento dell'informazione tramite procedure automatiche.



L'informatica descrive la codifica, l'analisi, la manipolazione e la trasmissione dell'informazione senza curarsi dello strumento di calcolo utilizzato, infatti, se un domani i calcolatori utilizzati saranno diversi da quelli attuali, i principi dell'informatica saranno sempre validi anche sui nuovi calcolatori utilizzati.

Oggigiorno abbiamo tipicamente 2 tipi di informatica ovvero, l'**informatica classica** e l'**informatica quantistica**.

L'**informatica quantistica**, a differenza della sua controparte classica, ha come unità fondamentale il qubit, si basa principalmente sulla matematica classica, meccanica quantistica e logica quantistica e fa uso principalmente di porte logiche quantistiche e circuiti logici quantistici.

Un **qubit**, a differenza del bit classico, può valere 0, 1 o entrambi i valori grazie al concetto di **sovrapposizione quantistica**.

Un qubit di solito viene rappresentato in questo modo:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

dove  $|\alpha|^2$  è la probabilità che il qubit vale 0 e  $|\beta|^2$  è la probabilità che il qubit vale 1. Abbiamo poi che:

$$|\alpha|^2 + |\beta|^2 = 1 \wedge \alpha, \beta \in \mathbb{C}$$

Avremo una sovrapposizione quantistica fino a quando non andremo ad eseguire una misura dopodiché il bit quantistico collassa in un bit classico valendo solamente 0 oppure 1.

Quindi:

$$|\Psi\rangle = \begin{cases} \alpha|0\rangle = |0\rangle = 0 & \text{se } |\alpha|^2 = 1 \\ \beta|1\rangle = |1\rangle = 1 & \text{se } |\beta|^2 = 1 \\ \alpha|0\rangle + \beta|1\rangle & \text{se } |\alpha|^2, |\beta|^2 \neq 1 \wedge |\alpha|^2 + |\beta|^2 = 1 \end{cases}$$

La notazione  $|\Phi\rangle$  prende il nome di ket, dove  $\Phi$  è uno stato quantistico formato da n qubit.

### Algoritmo quantistico:

Un algoritmo quantistico è progettato per essere eseguito su un computer quantistico sfruttando le proprietà della meccanica quantistica, utilizza come unità di informazione il qubit e i calcoli possono essere eseguiti in parallelo grazie all'entanglement e alla sovrapposizione quantistica. Questa capacità di eseguire i calcoli in parallelo

rende gli algoritmi quantistici potenzialmente molto più veloci per alcuni problemi specifici rispetto agli algoritmi classici.

**Ma perché un bit quantistico è "più potente" di un bit classico?** Per rispondere a questa domanda dobbiamo spiegare meglio la **sovrapposizione quantistica**, l'**entanglement** e l'**interferenza quantistica** che sono dei concetti che stanno alla base della meccanica quantistica. Questi concetti verranno trattati nella sezione 2.3 riguardante i concetti di meccanica quantistica. Prima però diamo un'occhiata ad alcuni concetti di matematica per l'informatica quantistica.

## 2.2 Concetti di matematica

L'informatica quantistica si basa principalmente su dei concetti di algebra lineare ed algebra tensoriale.

In questa sezione verrà fornita un elenco di alcuni concetti di algebra lineare con la spiegazione del loro utilizzo e, riporteremo la definizione del prodotto tensoriale con la spiegazione del suo utilizzo.

Inoltre, verrà riportata la definizione di spazio di Hilbert e la definizione del qubit e della porta logica quantistica da un punto di vista matematico.

### **Spazio di Hilbert:**

A differenza dello spazio tridimensionale che conosciamo tutti, lo spazio di Hilbert è uno spazio che ha infinite dimensioni. È un concetto matematico che permette di rappresentare tutte le possibili configurazioni di un sistema fisico. È particolarmente utile nella fisica quantistica per descrivere sistemi complessi.

### **Spazio di Hilbert applicato ai qubit:**

Lo stato di un qubit può essere rappresentato come un punto nello spazio di Hilbert. Questo significa che possiamo usare le proprietà matematiche dello spazio di Hilbert per descrivere e manipolare i qubit.

### **Spazio di Hilbert applicato alle porte logiche quantistiche:**

Le porte logiche quantistiche sono operazioni che modificano lo stato dei qubit. Esse possono essere rappresentate come delle trasformazioni nello spazio di Hilbert. Ad esempio, una porta logica quantistica può ruotare lo stato di un qubit attorno ad un asse specifico, cambiando la sua posizione nello spazio di Hilbert. Questo è fondamentale per eseguire calcoli quantistici, poiché, permette di manipolare i qubit in modi molto precisi.

**Definizione matematica di qubit:**

Un qubit possiamo vederlo come un vettore situato nello spazio di Hilbert i cui suoi elementi sono degli scalari appartenenti ai numeri complessi.

Un qubit può essere rappresentato tramite la notazione ket oppure tramite la notazione bra.

**Rappresentazione di un qubit tramite la notazione ket:**

$$|a\rangle = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = a_1|x_1\rangle + a_2|x_2\rangle + \dots + a_n|x_n\rangle$$

**Rappresentazione di un qubit tramite la notazione bra:**

$$\langle b| = \begin{bmatrix} b_1^* & b_2^* & \dots & b_n^* \end{bmatrix} = b_1^*\langle x_1| + b_2^*\langle x_2| + \dots + b_n^*\langle x_n|$$

**Definizione matematica di porta logica quantistica:**

Una porta logica quantistica non è altro che una matrice quadrata unitaria situata nello spazio di Hilbert i cui suoi elementi sono degli scalari appartenenti ai numeri complessi.

**Alcuni concetti di algebra lineare che stanno alla base dell'informatica quantistica:**

- **Trasposizione di un qubit:** questa operazione viene eseguita per ricavare il bra di un ket o viceversa.
- **coniugazione di un numero complesso:** questa operazione viene eseguita per ricavare il bra di un ket o viceversa.
- **Collegamento tra un ket ed il suo bra:** per ottenere il bra di un ket o viceversa si applica la trasposta del qubit e la coniugazione a tutti gli scalari del qubit ottenuto.
- **Scalare x qubit:** questa operazione possiamo trovarla per esempio quando dobbiamo eseguire una porta logica su un qubit.
- **Scalare x matrice di una porta logica quantistica:** questa operazione possiamo trovarla per esempio quando dobbiamo eseguire una porta logica su un qubit.

- **Moltiplicazione tra 2 matrici di 2 porte logiche quantistiche:** questa operazione possiamo trovarla per esempio quando dobbiamo definire una nuova porta logica quantistica.
- **Moltiplicazione porta logica quantistica x ket:** questa operazione possiamo trovarla quando dobbiamo applicare una porta logica quantistica ad un qubit (in questo caso un ket).
- **Moltiplicazione bra x porta logica quantistica:** questa operazione possiamo trovarla quando dobbiamo applicare una porta logica quantistica ad un qubit (in questo caso un bra).

### Operazione bra-ket:

L'operazione bra-ket si applica tra un bra e un ket e può essere vista come il prodotto scalare tra 2 vettori (1 bra e 1 ket) nello spazio di Hilbert. L'operazione è identica al prodotto scalare che conosciamo. L'operazione bra-ket viene indicata nel seguente modo:  $\langle a|b \rangle$  dove, appunto,  $a$  è un bra e  $b$  è un ket.

### Prodotto tensoriale:

Il prodotto tensoriale è una operazione dell'algebra tensoriale e viene utilizzata per esempio:

- tra 2 qubit per ottenere il qubit combinato (se abbiamo 2 ket o 2 bra) oppure una matrice (se abbiamo 1 ket e 1 bra o viceversa) oppure se dobbiamo definire una nuova porta logica quantistica.
- tra 1 qubit e 1 porta logica quantistica se dobbiamo definire una nuova porta logica quantistica.
- tra 2 porte logiche quantistiche se dobbiamo definire una nuova porta logica quantistica.

### Prodotto tensoriale tra n ket o n bra:

Il prodotto tensoriale tra n ket o n bra consiste nel creare un ket o bra formato da tutte le combinazioni degli elementi dei qubit di partenza sottoforma di moltiplicazione.

### Prodotto tensoriale tra 1 bra e 1 ket o viceversa:

Nel prodotto tensoriale tra 1 bra e 1 ket o viceversa, se il ket e il bra hanno n elementi, allora, otteniamo una matrice quadrata  $n \times n$  tale che: la riga i-esima è formata dal primo elemento del primo qubit che moltiplica per ogni cella un elemento del

secondo qubit (dal primo all'ultimo).

**Prodotto tensoriale tra un ket e una porta logica quantistica e viceversa:**

Per eseguire il prodotto tensoriale tra un ket e una porta logica quantistica e viceversa applichiamo le seguenti formule:

$$|a\rangle \otimes B = \begin{bmatrix} a_1 B \\ a_2 B \\ \vdots \\ a_n B \end{bmatrix}$$

$$B \otimes |a\rangle = \begin{bmatrix} b_{11}|a\rangle & b_{12}|a\rangle & \dots & b_{1n}|a\rangle \\ b_{21}|a\rangle & b_{22}|a\rangle & \dots & b_{2n}|a\rangle \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1}|a\rangle & b_{n2}|a\rangle & \dots & b_{nn}|a\rangle \end{bmatrix}$$

**Prodotto tensoriale tra un bra e una porta logica quantistica e viceversa:**

Per eseguire il prodotto tensoriale tra un bra e una porta logica quantistica e viceversa applichiamo le seguenti formule:

$$\langle a| \otimes B = \begin{bmatrix} a_1^* B & a_2^* B & \dots & a_n^* B \end{bmatrix}$$

$$B \otimes \langle a| = \begin{bmatrix} b_{11}\langle a| & b_{12}\langle a| & \dots & b_{1n}\langle a| \\ b_{21}\langle a| & b_{22}\langle a| & \dots & b_{2n}\langle a| \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1}\langle a| & b_{n2}\langle a| & \dots & b_{nn}\langle a| \end{bmatrix}$$

**Prodotto tensoriale tra 2 porte logiche quantistiche:**

Per eseguire il prodotto tensoriale tra 2 porte logiche quantistiche applichiamo la seguente formula:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \dots & a_{nn}B \end{bmatrix}$$

Anche se è sempre possibile prendere il prodotto tensore di due stati a qubit singolo per formare uno stato a due qubit, non tutti gli stati quantistici a due qubit possono

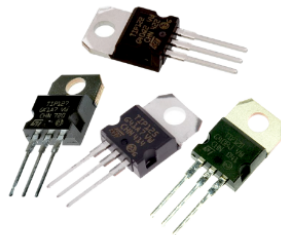
essere scritti come prodotto tensore di due stati a qubit singolo. Quando questo accade abbiamo che tale stato a due qubit, che non può essere scritto come prodotto tensoriale di stati a qubit singolo, viene chiamato "stato sottoposto ad entanglement": i due qubit sono definiti come sottoposti ad entanglement.

## 2.3 Concetti di meccanica quantistica

In questa sezione verranno riportati i concetti della meccanica quantistica che stanno alla base dell'informatica quantistica e ci aiuteranno anche a capire perché possiamo svolgere dei compiti più velocemente rispetto all'informatica classica.

### 2.3.1 Sovrapposizione quantistica

Nei componenti hardware classici, ogni bit corrisponde ad un transistor, e gli stati 0 e 1 corrispondono ai casi in cui il transistor è, rispettivamente, aperto o chiuso. Un transistor è formato da materiali semiconduttori che possono contenere milioni di atomi di silicio. Un transistor è, quindi, un oggetto macroscopico, le cui proprietà sono governate dalle leggi dell'elettromagnetismo classico introdotte da Maxwell nella metà del 1800.

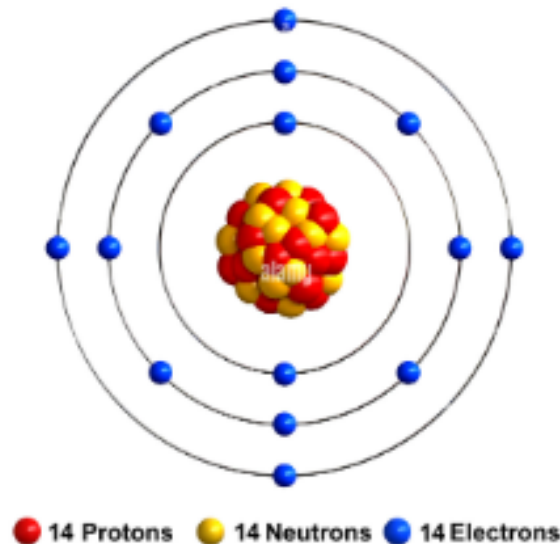


**Figura 2.1:** Transistor

Per costruire un qubit è necessario identificare gli oggetti che non seguono le leggi della fisica classica, ma quelle della meccanica quantistica. La meccanica quantistica è stata introdotta agli inizi del 1900 per spiegare delle osservazioni sperimentali che non potevano essere spiegate con le leggi della meccanica classica (le leggi di Newton) e le leggi dell'elettromagnetismo (le leggi di Maxwell). Gli esperimenti coinvolgevano i componenti elementari della materia come, ad esempio, singoli atomi o molecole. Atomi e molecole sono quindi i building blocks ideali per costruire

un computer quantistico.

Se per esempio considerassimo un atomo di silicio, esso ha un nucleo formato da 14 protoni con carica positiva e 14 neutroni con carica netta pari a 0 e attorno al nucleo ruotano 14 elettroni con carica negativa:

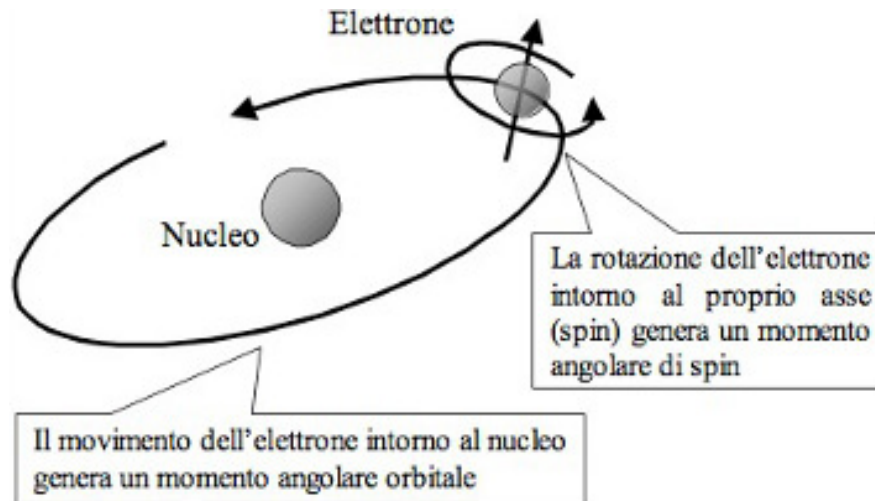


**Figura 2.2:** 1 atomo di silicio con 14 elettroni

Ora, consideriamo di utilizzare uno dei 14 elettroni come qubit, abbiamo bisogno di rappresentare gli stati 0 e 1 che per distinguerli dai bit classici utilizzeremo  $|0\rangle$  e  $|1\rangle$ . Per fare ciò utilizzeremo il concetto di spin:

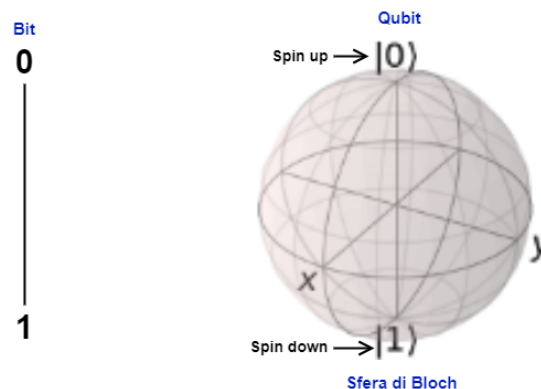
Lo spin è una proprietà fondamentale di un elettrone che è quantizzata, cioè può assumere solo due valori ovvero, lo stato "spin up" ( $|0\rangle$ ) e lo stato "spin down" ( $|1\rangle$ ). L'elettrone poi, durante il suo funzionamento, ruota attorno al nucleo dell'atomo avendo una certa probabilità di essere nello stato "spin up" ( $\rightarrow |0\rangle \rightarrow 0$ ) e una certa probabilità di essere nello stato "spin down" ( $\rightarrow |1\rangle \rightarrow 1$ ).

Quando invece andiamo ad eseguire una misura sull'elettrone (controlliamo la sua posizione nell'orbita) l'elettrone assumerà solamente o lo stato "spin up" ( $\rightarrow |0\rangle \rightarrow 0$ ) oppure lo stato "spin-down" ( $\rightarrow |1\rangle \rightarrow 1$ ).



**Figura 2.3:** Esempio di spin-up e spin-down

Possiamo quindi rappresentare il bit classico e il bit quantistico nei seguenti modi:



**Figura 2.4:** Bit e Qubit

### 2.3.2 Entanglement ed entangled

Due qubit distinti possono essere collegati tra loro in modo molto stretto, fino al punto, che indipendentemente dalla loro distanza il comportamento di uno può dipendere dal comportamento dell'altro e viceversa. In generale, dato un sistema a  $n$  qubit, se non è possibile decomporre lo stato totale nei qubit che lo compongono, allora questi sono detti entangled. In pratica, nessun qubit ha uno stato individuale, ma solo l'insieme di tutti i qubit ha uno stato ben definito.

Esempio:

Un esempio di 2 qubit sottoposti ad entanglement è lo stato di Bell o stato di Einstein-Podolsky-Rosen (EPR). Di seguito sono riportati 2 stati di Bell:



- $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$
- $|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

### 2.3.3 Interferenza quantistica

L'interferenza quantistica è una conseguenza della sovrapposizione quantistica. Gli stati dei qubit possono interferire tra loro perché ogni stato è descritto da un'ampiezza di probabilità, analoga alle ampiezze delle onde. Quindi, l'interferenza si verifica quando le onde quantistiche interagiscono tra loro, con conseguente interferenza costruttiva o distruttiva. L'interferenza quantistica svolge un ruolo cruciale in molte tecnologie quantistiche. Una delle applicazioni più promettenti è il calcolo quantistico, in cui i qubit si basano sui principi dell'interferenza quantistica per eseguire i calcoli. Un'altra applicazione è la crittografia quantistica, in cui viene utilizzata l'interferenza quantistica per garantire la sicurezza dei canali di comunicazione. Questi effetti vengono usati negli algoritmi del calcolo quantistico e li rendono fondamentalmente diversi dagli algoritmi classici. L'interferenza viene usata insieme all'entanglement per consentire l'accelerazione quantistica promessa dal calcolo quantistico.

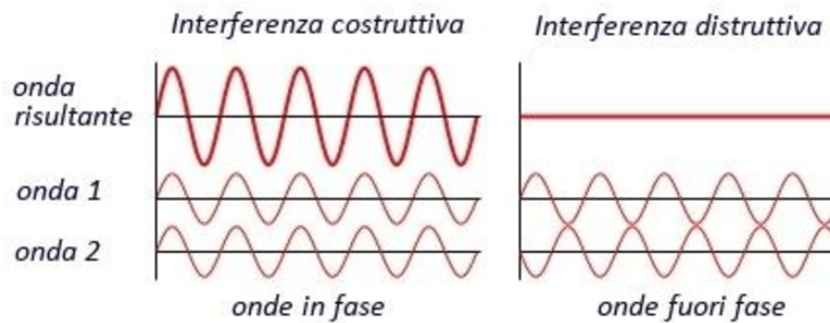
#### **Interferenza costruttiva:**

L'interferenza costruttiva si verifica quando due onde quantistiche si sommano per produrre un'onda più grande. In questo caso, la fase relativa tra le due onde è tale da rafforzarsi a vicenda. L'interferenza costruttiva migliora l'ampiezza.

#### **Interferenza distruttiva:**

L'interferenza distruttiva si verifica quando due onde quantistiche si annullano a vicenda. In questo caso, la fase relativa tra le due onde è tale da opporsi. L'interferenza distruttiva annulla l'ampiezza.

Esempio di interferenza quantistica:



**Figura 2.5:** Esempio di interferenza quantistica

## 2.4 Potenza del calcolo quantistico e numero di stati possibili

La potenza del calcolo quantistico si genera, in parte, perché le dimensioni dello spazio vettoriale dei vettori dello stato quantistico crescono in modo esponenziale con il numero di qubit. Ciò significa che, mentre un singolo qubit può essere modellato in modo semplice, la simulazione di un calcolo quantistico a 50 qubit probabilmente potrebbe superare i limiti dei supercomputer esistenti. L'aumento delle dimensioni del calcolo di un solo qubit aggiuntivo raddoppia la memoria necessaria per archiviare lo stato e raddoppia approssimativamente il tempo di calcolo. Questo rapido raddoppio della potenza di calcolo è il motivo per cui un computer quantistico con un numero relativamente ridotto di qubit può superare di gran lunga i supercomputer più potenti di oggi e di domani per alcune attività di calcolo. Per capire meglio l'enorme capacità dei computer quantistici, cerchiamo di capire quanti stati possibili abbiamo con  $n$  bit e con  $n$  qubit:

- $n$  bit  $\rightarrow 2^n$  possibili stati.
- $n$  qubit  $\rightarrow 2^n +$  tutte le possibili sovrapposizioni.

Possiamo già notare che, solamente con 3 qubit, abbiamo un enorme numero di stati possibili (infinite combinazioni).

## 2.5 Transpilazione quantistica

Per transpilazione quantistica intendiamo il processo in cui prendiamo un circuito quantistico scritto con un linguaggio di programmazione ad alto livello (come per esempio Qiskit e Q#) e lo trasformiamo in un formato compatibile per una specifica piattaforma hardware quantistica tramite un programma chiamato transpiler. Questo processo tiene conto dei vincoli hardware, come le limitazioni sui gate e i tempi di esecuzione, mantenendo le funzionalità del circuito originale, ovvero, il circuito quantistico di partenza è uguale al circuito quantistico finale anche se è scritto in maniera diversa. Il transpiler esegue una serie di trasformazioni per raggiungere questo obiettivo.

Qiskit offre 4 livelli di ottimizzazione, ovvero:

- **Livello 0:** nessuna ottimizzazione.
- **Livello 1:** ottimizzazione leggera.
- **Livello 2:** ottimizzazione pesante.
- **Livello 3:** ottimizzazione ancora più pesante.

Le transpilazioni considerate sono le seguenti 5:

Nome	Descrizione	Porte quantistiche utilizzate
original	Set di porte quantistiche impiegato nella letteratura originale sui quantum code smells.	U1, U2, U3, RZ, SX, X, CX, ID
ibm_perth	Set di porte quantistiche utilizzato dalla macchina quantistica IBM ibm_perth.	CX, ID, RZ, SX, X
ibm_sherbrooke	Set di porte quantistiche utilizzato dalla macchina quantistica IBM ibm_sherbrooke.	ECR, ID, RZ, SX, X
rpcx	Set di porte quantistiche universali noto composto dalla porta quantistica non controllata, dalle porte quantistiche di rotazione di tutti gli assi e dalla porta quantistica di fase.	CX, RX, RY, RZ, P
simple	Simile alla transpilazione rpcx ma basato sulla porta quantistica di rotazione parametrizzata U3, che può eseguire la rotazione su tutti gli assi e nella fase.	CX, U3

**Tabella 2.1:** Transpilazioni considerate

## 2.6 Porte logiche quantistiche

Le porte logiche quantistiche, a differenza delle porte logiche classiche, possono essere potenzialmente infinite. Siccome le porte logiche quantistiche sono molte di più rispetto alle porte logiche classiche allora mi limiterò a riportarne solamente 5 utilizzate nelle varie transpilazioni. Per capire come è fatta la matrice di un'altra qualsiasi porta logica quantistica possiamo recarci sul sito web di **qiskit** il cui link è riportato nella bibliografia e ricercare il nome della porta logica quantistica (Gate) a cui siamo interessati.

Quindi, le porte logiche quantistiche che considereremo sono le seguenti 5:

X, Z, CX, CZ, H.




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

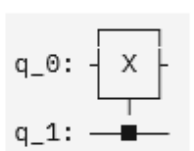
**Figura 2.6:** Gate X



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

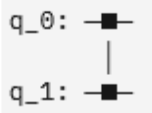
**Figura 2.7:** Gate Z



$$CX_{q_0, q_1} = I \otimes |0\rangle\langle 0| + X \otimes |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$
  


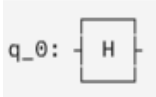
$$CX_{q_1, q_0} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

**Figura 2.8:** Gate CX



$$CZ_{q_0, q_1} = I \otimes |0\rangle\langle 0| + Z \otimes |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Figura 2.9: Gate CZ



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

Figura 2.10: Gate H

L'uso di più porte logiche quantistiche forma un **circuito logico quantistico**.  
Esempio circuito logico quantistico:

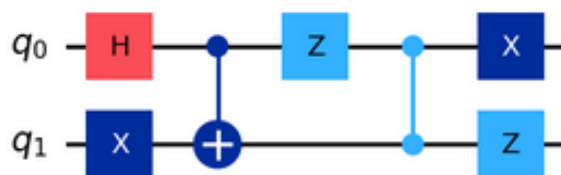


Figura 2.11: Esempio di circuito logico quantistico

## 2.7 Code Smells

Un code smell, nell'ingegneria del software, indica una serie di caratteristiche che il codice sorgente può avere e che sono generalmente riconosciute come probabili indicazioni di un difetto di programmazione. Non sono e non rivelano dei "bug" ma, sono debolezze di progettazione che riducono la qualità del software, a prescindere dall'effettiva correttezza del suo funzionamento.

i code smells presenti in un sistema quantistico si chiamano: **Quantum Code Smells**. Durante il lavoro di tirocinio interno ho dovuto progettare ed implementare una web application chiamata **SearchQS** che ha proprio il compito di individuare dei Quantum Code Smells in un sistema quantistico inserito in input.

Alcuni Quantum Code Smells sono individuati con una analisi statica ed altri con una analisi dinamica:

- **Analisi statica:** l'analisi statica prende in input un file Python con del codice scritto in Qiskit e, successivamente, tramite Python AST viene analizzato il file per individuare in quale quantità ogni quantum code smell è presente.
- **Analisi dinamica:** l'analisi dinamica prende in input un file Python con del codice scritto in Qiskit e, successivamente, crea una matrice ( $n \times m$ ) per ogni circuito presente nel file. Ogni matrice è formata nel seguente modo:
  - la cella  $(0,0) = \text{None}$ .
  - le celle  $(0,1), (0,2), \dots, (0,m)$  della prima riga sono formati da numeri naturali sequenziali da 1 a  $m$  rappresentanti i time-stamps della matrice.
  - le celle  $(1,0), (2,0), \dots, (t,0)$  della prima colonna sono uguali rispettivamente a  $'qb-0', 'qb-1', \dots, 'qb-(t-1)$  e rappresentano i qubit del circuito quantistico mentre, le celle  $(t+1,0), (t+2,0), \dots, (n,0)$  sempre della prima colonna sono uguali rispettivamente a  $'cb-0', 'cb-1', \dots, 'cb-(k-1)'$  e rappresentano i bit classici del circuito quantistico. Inoltre,  $t + k = n$ .
  - le restanti celle, ovvero, le celle nelle posizioni  $ij$  dove  $i$  è la riga  $i$ -esima e  $j$  è la colonna  $j$ -esima e  $(i \geq 1 \wedge i \leq n)$  e  $(j \geq 1 \wedge j \leq m)$  possono essere uguali ad 1 delle seguenti stringhe:
    - \*  $' '$  se il bit classico o qubit  $i$  per il time-stamp  $j$  non ha una porta quantistica associata.
    - \*  $'nome\_porta\_quantistica(parametri)'$  oppure  $'nome\_porta\_quantistica()'$  se il bit classico o qubit  $i$  per il time-stamp  $j$  ha la porta logica quantistica  $'nome\_porta\_quantistica'$  associata.

Quando viene controllato in quale quantità è presente un quantum code smell allora, se otteniamo un valore  $= 0$  allora esso non è presente nel file o in un circuito altrimenti se otteniamo un valore  $> 0$  allora esso è presente nel file o in un circuito.

I quantum code smells analizzati da SearchQS sono in totale 8 e sono i seguenti:

Nome	Descrizione	Pratica migliore
CG (use of Customized Gates)	Ogni gate personalizzato è scomponibile in operatori framework incorporati. Questa scomposizione richiede un numero di operatori sostanzialmente più elevato rispetto alla soluzione equivalente realizzata esclusivamente con operatori incorporati.	Ottenere un circuito da eseguire su hardware.
ROC (Repeated set of Operations of Circuit)	A causa di limitazioni tecnologiche e fisiche, il numero di operazioni che si possono passare a un computer quantistico è limitato; pertanto, il circuito che implementa l'intero algoritmo dovrebbe essere preparato in modo tale che il numero di operazioni ripetute in sequenza da eseguire sia il minimo possibile.	Utilizzo del funzionamento del circuito per ridurre le dimensioni del circuito



Nome	Descrizione	Pratica migliore
NC (Non-parametrized Circuit)	I dispositivi reali funzionano in una politica condivisa. Il circuito dovrebbe essere progettato in modo parametrico per fornire simultaneamente i diversi valori iniziali, evitare di metterne in coda altri diversi e ridurre i payload di comunicazione.	Utilizzare le sweeps quando possibile.
LC (Long Circuit)	Le porte e le misurazioni unitarie sono soggette a errori (specialmente a causa del rumore quantistico). Maggiore è la profondità del circuito e/o più ampio è il circuito, maggiore è la probabilità di influenzare il comportamento previsto di un circuito quantistico.	Profondità del cancello corto.

Nome	Descrizione	Pratica migliore
IM (Intermediate Measurement)	Le misurazioni influenzano lo stato dell'intero sistema, rendendolo soggetto a più errori. Dovrebbero essere posticipate all'ultima operazione sul circuito per evitare la propagazione degli errori.	Misure terminali.
IdQ (Idle Qubit)	Con la tecnologia attuale, è possibile garantire la correttezza di uno stato solo per periodi molto brevi. I qubit inattivi per troppo tempo aumentano la perdita di informazioni quantistiche che possono compromettere i risultati di un circuito quantistico.	Mantenere occupati i Qubit.

Nome	Descrizione	Pratica migliore
IQ (Initialization of Qubit)	Mantenere la coerenza di uno stato eccitato quantistico è tecnologicamente difficile. Quindi, inizialmente, si dovrebbe mantenerlo nel suo stato fondamentale (vale a dire, nello stato $ 0\rangle$ ) il più a lungo possibile.	Inizializzazione ritardata dei qubit.
LPQ (no-alignment between the Logical and Physical Qubits)	La topologia dei qubit reali ha un impatto sul comportamento del circuito, ovvero i risultati ottenuti dal circuito possono cambiare in base alla configurazione fisica dei qubit. Non allineare i qubit logici ai qubit fisici appropriati può portare a risultati meno accurati.	Selezione di qubit

**Tabella 2.2:** Circuiti quantistici considerati

## 2.8 QSmell e SearchQS

Come riportato dal titolo della tesi, la Piattaforma Web SearchQS si occupa dell'individuazione ed esplorazione di problemi di progettazione in circuiti quantistici. Uno di questi problemi è l'individuazione dei quantum code smells. Quindi, SearchQS si

occuperà dell'individuazione di questi ultimi. In particolare, verranno considerati in totale 8 quantum code smells diversi di cui: 6 si possono trovare nei circuiti di un file analizzato e 2 si possono trovare nella struttura sintattica di un file analizzato.

Prima della creazione di SearchQS, esisteva ed esiste ancora un sistema che si occupa di analizzare un sistema quantistico per individuare dei quantum code smells in esse. L'unica cosa è che non considera la possibilità di gestire un database per poter inserire i risultati delle analisi ottenute. Questo comporta la perdita dei risultati ottenuti.

SearchQS, a differenza di QSmell, è una web application che, oltre ad eseguire delle analisi, fornisce anche il loro salvataggio in un database in maniera tale da poterli visionare anche in un secondo momento e, quando non ci interessano più, possiamo tranquillamente eliminarli dal database. In SearchQS, un utente, per poter eseguire delle analisi, ha bisogno prima di registrarsi alla piattaforma e poi successivamente procedere con le analisi.

Nel database, quindi, verranno salvati, per ogni sistema analizzato:

- Il codice dei file Python analizzati.
- Le matrici generate per ogni analisi eseguita (sia con e sia senza transpilazione).
- I risultati ottenuti per ogni analisi statica e dinamica eseguita (sia con e sia senza transpilazione).

Inoltre, il sistema QSmell, ha contribuito all'apprendimento su come vengono individuati gli 8 quantum code smells considerati.

---

### **SearchQS: Una Piattaforma Web per l'Individuazione ed Esplorazione di Problemi di Progettazione in Circuiti Quantistici.**

---

In questo capitolo parleremo della web application SearchQS che ha come scopo l'individuazione ed esplorazione di problemi di progettazione in circuiti quantistici, uno di questi è l'individuazione dei quantum code smells. Quindi, SearchQS si occuperà dell'individuazione di questi ultimi. In particolare, dato un file da analizzare: verranno considerati in totale 8 quantum code smells diversi di cui:

- 6 verranno individuati nei circuiti quantistici di un file analizzato (analisi dinamica).
- 2 si possono trovare nella struttura sintattica di un file analizzato (analisi statica).

Prima della spiegazione di SearchQS, individuiamo delle domande di ricerca e rispondiamo ad esse.

**Domanda numero 1: Perché abbiamo bisogno di individuare dei quantum code smells presenti nei sistemi quantistici?**

Come spiegato nel capitolo 2, un code smell non è né un bug e né un modo per individuare i bug ma è un pezzo di codice che può essere migliorato, quindi, se

riuscissimo ad individuare i quantum code smells presenti in un file oppure in un circuito quantistico di un sistema quantistico allora potremmo capire dove modificare il codice per renderlo migliore.

**Domanda numero 2: Perché abbiamo bisogno di una web application per individuare i quantum code smells presenti in un sistema quantistico inserito in input?**

Se non avessimo un procedimento automatico per generare le matrici dei vari circuiti quantistici presenti in un file allora dovremmo crearle noi a mano e, siccome molto spesso potremmo incontrare dei circuiti quantistici molto grandi allora dovremmo generare manualmente delle matrici molto grandi e siccome in un file possono esserci molti circuiti quantistici allora dovremmo creare molte matrici rendendo il processo manuale ingestibile in termini di tempo e aumenterebbe il rischio di creare delle matrici sbagliate e, siccome 6 quantum code smells su 8 vengono individuati nelle matrici allora, se avessimo delle matrici errate allora, di conseguenza, avremmo dei risultati delle analisi errate. Inoltre, potrebbe essere utile salvare in un database i vari file analizzati con le varie matrici generate automaticamente e i risultati delle analisi effettuate. Ecco perché abbiamo bisogno di una web application in cui un utente si registra e dopo la registrazione può eseguire delle analisi su un sistema quantistico e successivamente i risultati verranno salvati in un database per poter accedere ad essi in qualsiasi momento.

Di seguito, saranno riportate le spiegazioni dei metodi e delle funzioni più importanti che si occupano dell'analisi di un sistema quantistico e su come vengono individuati i vari quantum code smells.

Per risalire al codice visitare il repository github di SearchQS riportato nella bibliografia.

**Metodo `execution_analyses` file `analysis_service_impl.py`:**

Il metodo `execution_analyses` inizia dichiarando diverse variabili e creando una connessione al database MySQL. Vengono inizializzati diversi DAO per gestire le operazioni sui dati e viene avviata una transazione. Gli attributi passati al metodo vengono controllati e, se ci sono errori, viene sollevata un'eccezione. I file selezionati vengono letti e il loro contenuto viene memorizzato in un dizionario. Per ogni file, viene creato un oggetto `SourceFile`, il cui contenuto viene impostato dal codice letto. Se il file non esiste nel database, viene creato. Vengono poi ottenute le chiamate di funzione e i nomi dei circuiti quantistici dal file. Viene creata un'analisi per ogni transpilazione selezionata e, se la transpilazione è `None`, non viene eseguita nessuna transpilazione. Per ogni file sorgente, vengono eseguite un'analisi statica e dinamica e i risultati vengono memorizzati nel database. Se tutto va bene, la transazione viene confermata e i dati vengono salvati nel database. Se si verifica un'eccezione, viene stampato un messaggio di errore, la transazione viene annullata e vengono restituiti gli errori rilevati. Infine, la connessione al database viene chiusa.

**Funzione `get_matrix` file `analysis_operation.py`:**

La funzione `get_matrix` carica dinamicamente un modulo Python da un file specificato, estrae un circuito quantistico dal modulo, crea una matrice che rappresenta il circuito e restituisce una lista contenente questa matrice.

Se c'è un errore durante il caricamento del modulo allora stampa un messaggio di errore.

**Funzione `make_matrix` file `analysis_operation.py`:**

La funzione `make_matrix` crea una matrice che rappresenta un circuito quantistico. La matrice include informazioni sui qubit, i bit classici e le operazioni eseguite nel circuito. Ogni colonna rappresenta un'operazione nel circuito, mentre le righe rappresentano i qubit e i bit classici. La matrice viene popolata con i nomi delle operazioni eseguite su ciascun qubit in ogni colonna.

**Funzione `get_result_static_analysis` file `analysis_operation.py`:**

Questa funzione esegue un'analisi statica su un file sorgente Python contenente del codice Qiskit. Se specificato, aggiunge del codice di transpilazione al file. Poi, legge e analizza l'albero sintattico del file, esegue dei controlli per verificare se siano presenti i quantum code smells LPQ e NC e restituisce i risultati ottenuti.

**Funzione `get_results_dynamic_analysis` file `analysis_operation.py`:**

Questa funzione esegue un'analisi dinamica su un circuito quantistico di un file

sorgente Python contenente codice Qiskit. Se specificato, aggiunge codice di traspilazione al file. Poi, per ogni circuito quantistico, crea una matrice che rappresenta il circuito, esegue dei controlli per verificare se siano presenti i quantum code smells IdQ, IQ, IM, LC, ROC e CG e restituisce i risultati ottenuti con l'analisi dinamica in una lista di oggetti entity ResultDynamicAnalysis chiamata results\_da.

**Come viene individuato il quantum code smell LPQ:**

Analizziamo l'albero di sintassi astratta (AST) del file Python da analizzare per contare quante volte la funzione transpile viene chiamata, senza utilizzare la parola chiave initial\_layout. Ecco come funziona:

- Inizializziamo un contatore num\_transpiles\_without\_initial\_layout a 0.
- Scorriamo tutti i nodi dell'albero AST. Se il nodo è una chiamata di funzione (ast.Call):
  - Controlliamo se la funzione chiamata è transpile.
  - Verifichiamo se la parola chiave initial\_layout non è utilizzata.
- Se initial\_layout non è utilizzata allora incrementiamo il contatore.

Alla fine, assegnamo il valore del contatore a self.value e lo restituiamo.



**Come viene individuato il quantum code smell NC:**

Analizziamo l'albero di sintassi astratta (AST) del file Python da analizzare per contare quante volte le funzioni `execute` e `run` vengono chiamate, con particolare attenzione a quelle chiamate all'interno di loop. Inoltre, contiamo quante volte viene chiamata la funzione `bind_parameters`. Ecco come funziona:

1. Identifichiamo le linee dei loop:
  - scorriamo tutti i nodi dell'AST.
  - se il nodo è un loop allora aggiungiamo le linee di codice del loop a una lista `loops_lines`.
2. Conteggiamo delle chiamate a `execute` e `run`:
  - inizializziamo un contatore `num_executions` a 0.
  - scorriamo tutti i nodi dell'AST.
  - se il nodo è una chiamata di funzione (`ast.Call`):
    - controlliamo se la funzione chiamata è `execute` o `run`.
    - incrementiamo il contatore `num_executions` per ogni chiamata trovata.
  - se la chiamata si trova all'interno di un loop allora incrementiamo ulteriormente il contatore.
3. Conteggiamo le chiamate a `bind_parameters`:
  - inizializziamo un contatore `num_bind_parameters` a 0.
  - scorriamo tutti i nodi dell'AST.
  - se il nodo è una chiamata di funzione (`ast.Call`) e la funzione chiamata è `bind_parameters` allora incrementiamo il contatore `num_bind_parameters`.
4. Calcolo del risultato:
  - inizializziamo `self.value` a 0.
  - se il numero di chiamate a `execute` e `run` è maggiore del numero di chiamate a `bind_parameters` allora assegniamo a `self.value` la differenza tra `num_executions` e `num_bind_parameters`.

**Come viene individuato il quantum code smell IdQ:**

Analizziamo una matrice che rappresenta un circuito quantistico. Contiamo il numero massimo di operazioni consecutive tra due operazioni non vuote per ciascun qubit. Ignoriamo le operazioni di tipo "barrier". Restituiamo il valore massimo trovato fra tutte le operazioni consecutive.

**Come viene individuato il quantum code smell IQ:**

Analizziamo una matrice che rappresenta un circuito quantistico. Contiamo il numero massimo di operazioni tra l'inizializzazione di un qubit e il suo primo utilizzo. Ignoriamo le operazioni di tipo "barrier". Restituiamo il valore massimo trovato fra tutte le operazioni tra l'inizializzazione e il primo utilizzo del qubit.

**Come viene individuato il quantum code smell IM:**

Analizziamo una matrice che rappresenta un circuito quantistico. Contiamo il numero di operazioni di misura che non si trovano alla fine del circuito per ciascun qubit. Se un'operazione di misura è seguita da un'altra operazione allora viene conteggiata. Restituiamo il numero totale di queste operazioni di misura non finali.

**Come viene individuato il quantum code smell LC:**

Analizziamo una matrice che rappresenta un circuito quantistico. Contiamo il numero massimo di operazioni per qubit e per timestamp, ignorando le operazioni di tipo "barrier". Calcoliamo il prodotto tra il massimo numero di operazioni per qubit e il massimo numero di operazioni per timestamp e restituiamo questo valore.

**Come viene individuato il quantum code smell ROC:**

Analizziamo una matrice per trovare e contare le ripetizioni di sequenze di operazioni. Ecco come funziona:

1. Inizializzazione:

- impostiamo self.value a 0.
- otteniamo il numero di righe (num\_rows) e colonne (num\_columns) della matrice.
- inizializziamo num\_repetitions a 0 e time\_stamp a 1.

2. Ciclo principale:

- continuiamo ad iterare finché time\_stamp è minore del numero di colonne.  
Inizializziamo max\_slice\_size\_with\_a\_match a 0.

3. Ricerca delle sequenze:

- per ogni dimensione di sequenza (`slice_size`), creiamo due liste: `slice` e `next_slice`.
- popoliamo `slice` con le operazioni della matrice a partire da `time_stamp`.
- popoliamo `next_slice` con le operazioni della matrice a partire da `time_stamp + slice_size + 1`.
- se `slice` e `next_slice` sono uguali allora incrementiamo `num_repetitions` e aggiorniamo `max_slice_size_with_a_match`.

4. Aggiornamento del `time_stamp`:

- Se è stata trovata una sequenza corrispondente allora incrementiamo `time_stamp` di `max_slice_size_with_a_match`, altrimenti, incrementiamo `time_stamp` di 1.

Alla fine, assegniamo `num_repetitions` a `self.value` e lo restituiamo.

**Come viene individuato il quantum code smell CG:**

Analizziamo una matrice che rappresenta un circuito quantistico. Contiamo il numero di operazioni unitarie eseguite sui qubit, incrementando un valore ogni volta che trova un'operazione unitaria. Restituiamo il valore totale delle operazioni unitarie trovate.

---

### Analisi di alcuni sistemi quantistici

---

In questo capitolo analizzeremo 3 sistemi quantistici, ovvero:

- Sistema intitolato: VariAlgoritmi.
- Sistema intitolato: AlgoritmiAI.
- Sistema intitolato: AlgoritmiSicurezza.

I vari sistemi sono formati da file di tipo Python (.py) e file di tipo Jupiter Notebook (.ipynb) rappresentanti gli algoritmi sia nella loro versione classica e sia nella loro versione quantistica.

Di seguito saranno riportati i circuiti quantistici e le matrici dei file che verranno analizzati. Per visualizzare ed eseguire il codice dei vari sistemi visitare le repository github che ho caricato presenti nella bibliografia. I vari algoritmi quantistici sono stati ricavati da github, e da internet e sono stati rivisti e talvolta aggiornati e modificati per adattarli alla versione di Qiskit utilizzata e per i nostri scopi.

#### 4.1 Sistema VariAlgoritmi

In questo sistema sono presenti i seguenti file .py (in ordine alfabetico):

- Clonazione1Frase.py

- NumeroCasuale.py
- NumeroPseudoCasuale.py
- Teletrasporto1Frase.py
- Teletrasporto1Qubit.py
- utils.py

Circuiti analizzati:

- Circuito teleportation\_circuit presente nel file Teletrasporto1Qubit.py.
- Circuito random\_binary\_number\_circuit presente nel file NumeroCasuale.py

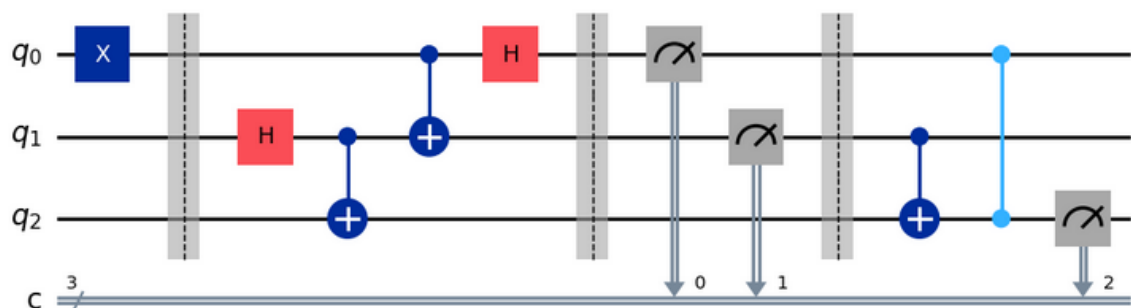
### File utils.py

Il file utils.py è un modulo Python contenente le funzioni utili per gli altri file Python presenti nel sistema.

### Clonazione e teletrasporto:

In informatica classica, se volessimo clonare un bit possiamo farlo, mentre, in informatica quantistica non possiamo farlo per il principio di no-cloning. Entra in gioco quindi il teletrasporto quantistico che ci permette di collegare 2 qubit a e b tramite il concetto di entanglement grazie alla porta quantistica Hadamard (H) e, successivamente, spostare l'informazione dal qubit a al qubit b eliminando l'informazione del qubit di partenza.

### Circuito teleportation\_circuit senza transpilazione:



**Figura 4.1:** Circuito teleportation\_circuit senza transpilazione

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13
qb-0	x()	barrier()	.	.	cx()	h()	barrier()	measure()	.	.	barrier()	.	.
qb-1	.	barrier()	h()	cx()	cx()	.	barrier()	.	measure()	barrier()	cx()	.	.
qb-2	.	barrier()	.	cx()	.	.	barrier()	.	.	barrier()	cx()	cz()	measure()
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.2: Matrice teleportation\_circuit senza transpilazione

Circuito teleportation\_circuit con transpilazione original:

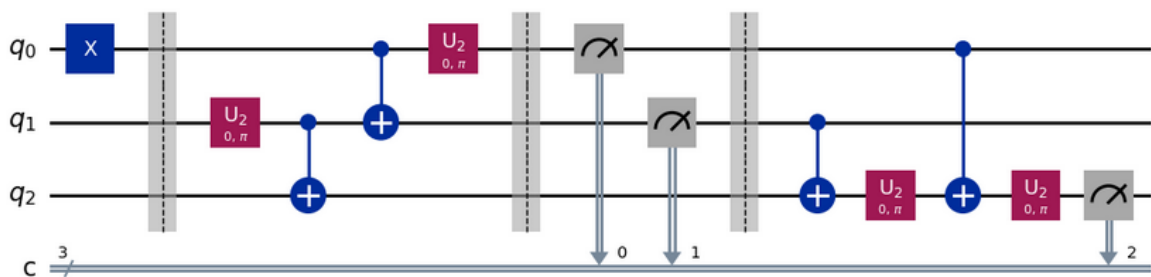


Figura 4.3: Circuito teleportation\_circuit original

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
qb-0	x()	barrier()	.	.	cx()	u2(int,float)	barrier()	measure()	.	barrier()	.	.	cx()	.	.
qb-1	.	barrier()	u2(int,float)	cx()	cx()	.	barrier()	.	measure()	barrier()	cx()	.	.	.	.
qb-2	.	barrier()	.	cx()	.	.	barrier()	.	.	barrier()	cx()	u2(int,float)	cx()	u2(int,float)	measure()
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.4: Matrice teleportation\_circuit original

Circuito teleportation\_circuit con transpilazione rpcx:

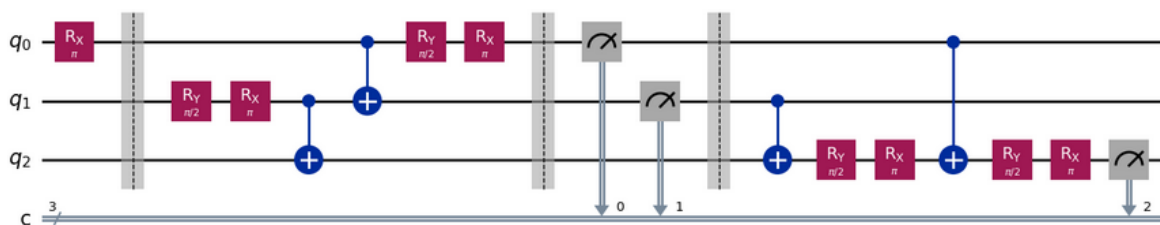


Figura 4.5: Circuito teleportation\_circuit rpcx

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
qb-0	rx(float)	barrier()	.	.	.	cx()	ry(float)	rx(float)	barrier()	measure()	.	.	barrier()	.	.	cx()	.	.	.
qb-1	.	.	barrier()	ry(float)	rx(float)	cx()	cx()	.	.	.	measure()	.	barrier()	cx()	.	.	.	.	.
qb-2	.	.	barrier()	.	.	cx()	.	.	.	.	.	.	barrier()	cx()	ry(float)	rx(float)	cx()	ry(float)	rx(float)
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.6: Matrice teleportation\_circuit rpcx

Circuito teleportation\_circuit con transpilazione simple:

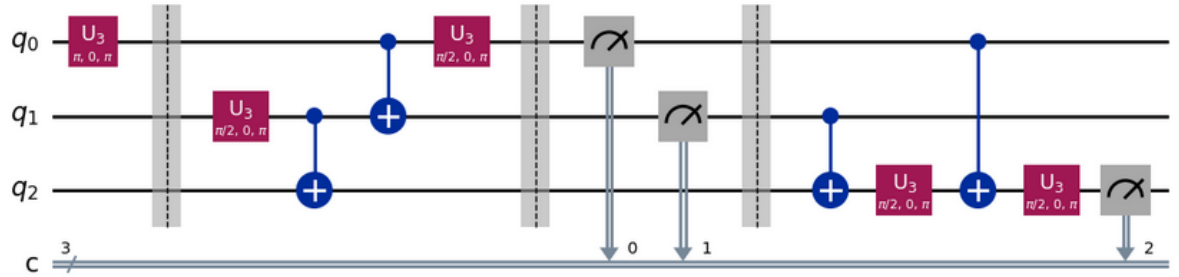


Figura 4.7: Circuito teleportation\_circuit simple

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
qb-0	u3(float,int,float)	barrier()	.	.	.	cx()	u3(float,int,float)	barrier()	measure()	.	.	barrier()	.	.	.
qb-1	.	.	barrier()	u3(float,int,float)	cx()	cx()	.	.	.	measure()	barrier()	cx()	.	.	.
qb-2	.	.	barrier()	.	.	.	.	.	.	.	barrier()	cx()	u3(float,int,float)	cx()	u3(float,int,float)
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.8: Matrice teleportation\_circuit simple

Circuito teleportation\_circuit con transpilazione ibm\_perth:

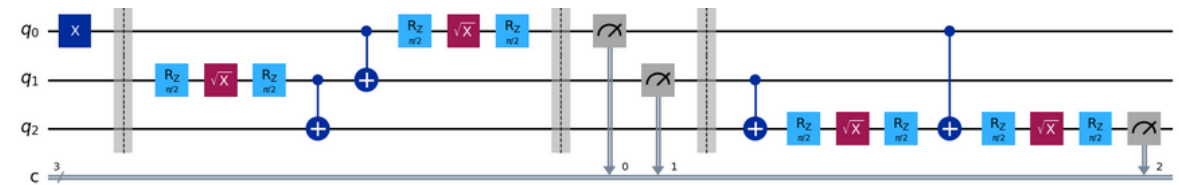


Figura 4.9: Circuito teleportation\_circuit ibm\_perth

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
qb-0	x()	barrier()	.	.	.	.	cx()	rz(float)	sx()	rz(float)	barrier()	measure()	.	.	barrier()	.	.	.	.	cx()	.	.	.
qb-1	.	.	barrier()	rz(float)	sx()	rz(float)	cx()	cx()	.	.	.	.	measure()	.	barrier()	cx()	.	.	.	.	.	.	.
qb-2	.	.	barrier()	.	.	.	cx()	.	.	.	.	.	.	.	barrier()	cx()	rz(float)	sx()	rz(float)	cx()	rz(float)	sx()	rz(float)
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.10: Matrice teleportation\_circuit ibm\_perth

### Circuito teleportation\_circuit con transpilazione ibm\_sherbrooke:

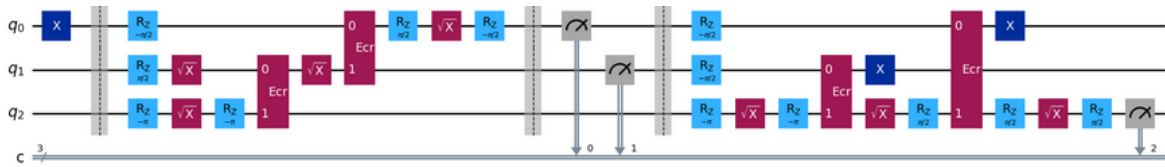


Figura 4.11: Circuito teleportation\_circuit ibm\_sherbroke

### Matrice associata:

Note	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
qb-0	x()			rz(float)						ecr()	rz(float)	sv()	rz(float)		barrier()	measure()		barrier()	rz(float)					ecr()				
qb-1		barrier()			sv()					ecr()	sv()	ecr()												ecr()	x()			
qb-2						rz(float)	sv()																					
cb-0																												
cb-1																												
cb-2																												
qb-0																												
qb-1																												
qb-2																												
cb-0																												
cb-1																												
cb-2																												

Figura 4.12: Matrice teleportation\_circuit ibm\_sherbroke

### Numero casuale e numero pseudo-casuale:

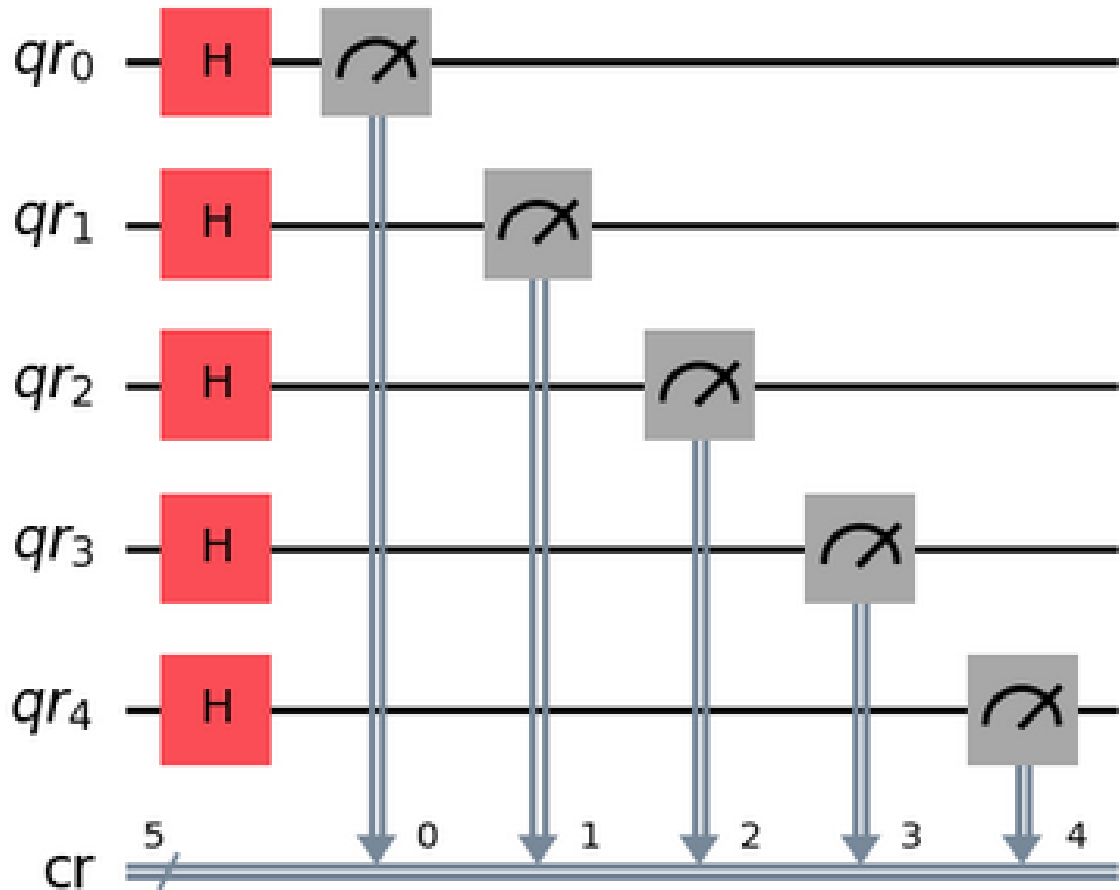
In informatica classica, quando noi vogliamo generare un numero casuale esso non potrà mai essere effettivamente casuale infatti, avremo sempre un numero pseudo-casuale. Un algoritmo classico non può generare numeri veramente casuali perché è deterministico per natura, il che significa che produce gli stessi output se viene eseguito con gli stessi input. Questo è il motivo per cui i numeri generati da un algoritmo classico sono chiamati “pseudo-casuali”. Anche se i numeri sembrano casuali e non mostrano alcun pattern evidente, se conosciamo l’input (chiamato “seme”) e l’algoritmo, possiamo prevedere l’output. Quindi, in teoria, non sono veramente casuali. Si cerca quindi di renderli il più possibile casuali utilizzando come seme per esempio l’ora in cui viene eseguito l’algoritmo (ora : minuti : secondi : millisecondi) oppure degli hardware che utilizzano dei processi fisici classici come il concetto di **moto browniano** ovvero, il fenomeno in cui si verifica quando in un fluido troviamo delle piccole particelle che si muovono in modo disordinato e casualmente in un fluido scontrandosi tra di loro però, purtroppo, non generano numeri casuali in maniera soddisfacente ed a volte possono essere anche prevedibili quindi, non potranno mai essere veramente casuali.

Questo discorso cambia con gli algoritmi quantistici, infatti, generare numeri casuali su un computer quantistico non è una cosa difficile in quanto l’impredicibilità della misura di un qubit in stato di sovrapposizione è di fatto una forma di casualità natu-



rile, quindi, quando andiamo ad eseguire una misura, la sovrapposizione collassa ottenendo un numero veramente casuale e non pseudo-casuale.

**Circuito random\_binary\_number\_circuit senza transpilazione:**



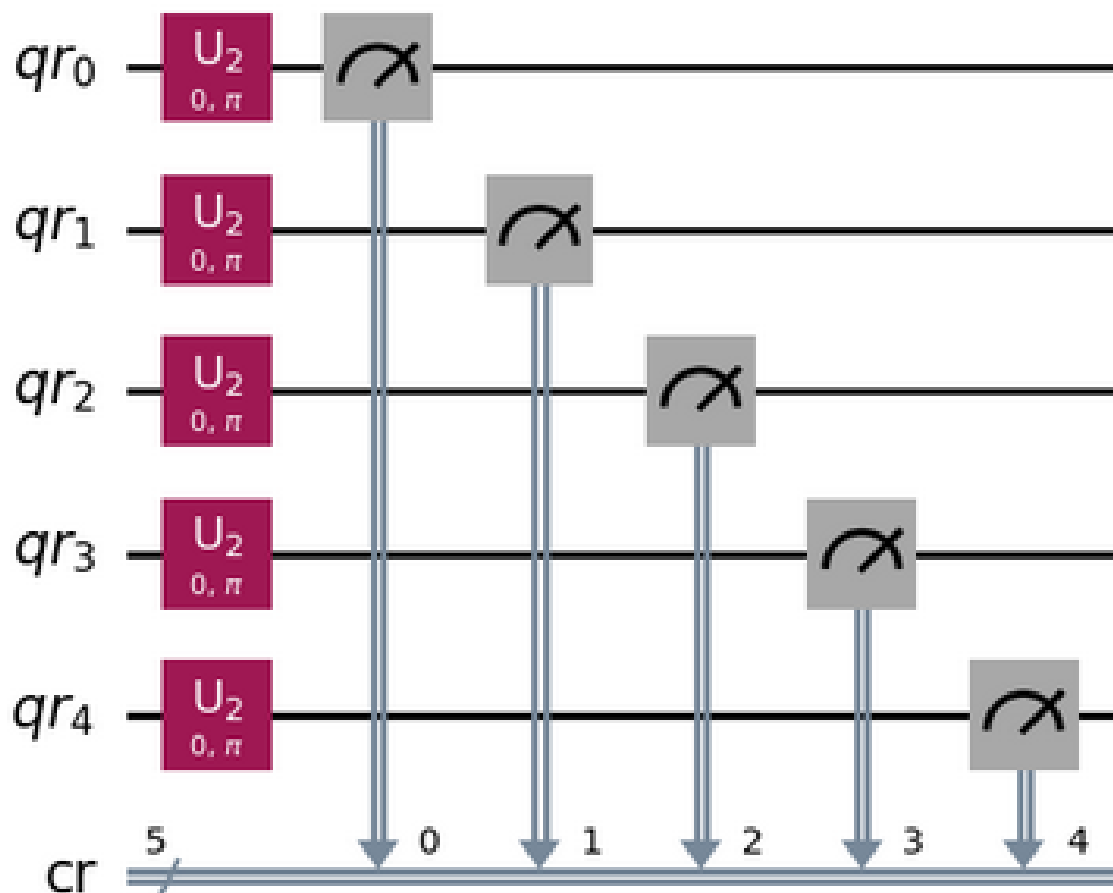
**Figura 4.13:** Circuito random\_binary\_number\_circuit senza transpilazione

**Matrice associata:**

None	1	2	3	4	5	6	7	8	9	10
qb-0	h()	'	'	'	'	measure()	'	'	'	'
qb-1	'	h()	'	'	'	'	measure()	'	'	'
qb-2	'	'	h()	'	'	'	'	measure()	'	'
qb-3	'	'	'	h()	'	'	'	'	measure()	'
qb-4	'	'	'	'	h()	'	'	'	'	measure()
cb-0	'	'	'	'	'	'	'	'	'	'
cb-1	'	'	'	'	'	'	'	'	'	'
cb-2	'	'	'	'	'	'	'	'	'	'
cb-3	'	'	'	'	'	'	'	'	'	'
cb-4	'	'	'	'	'	'	'	'	'	'

**Figura 4.14:** Matrice random\_binary\_number\_circuit senza transpilazione

Circuito random\_binary\_number\_circuit con transpilazione original:



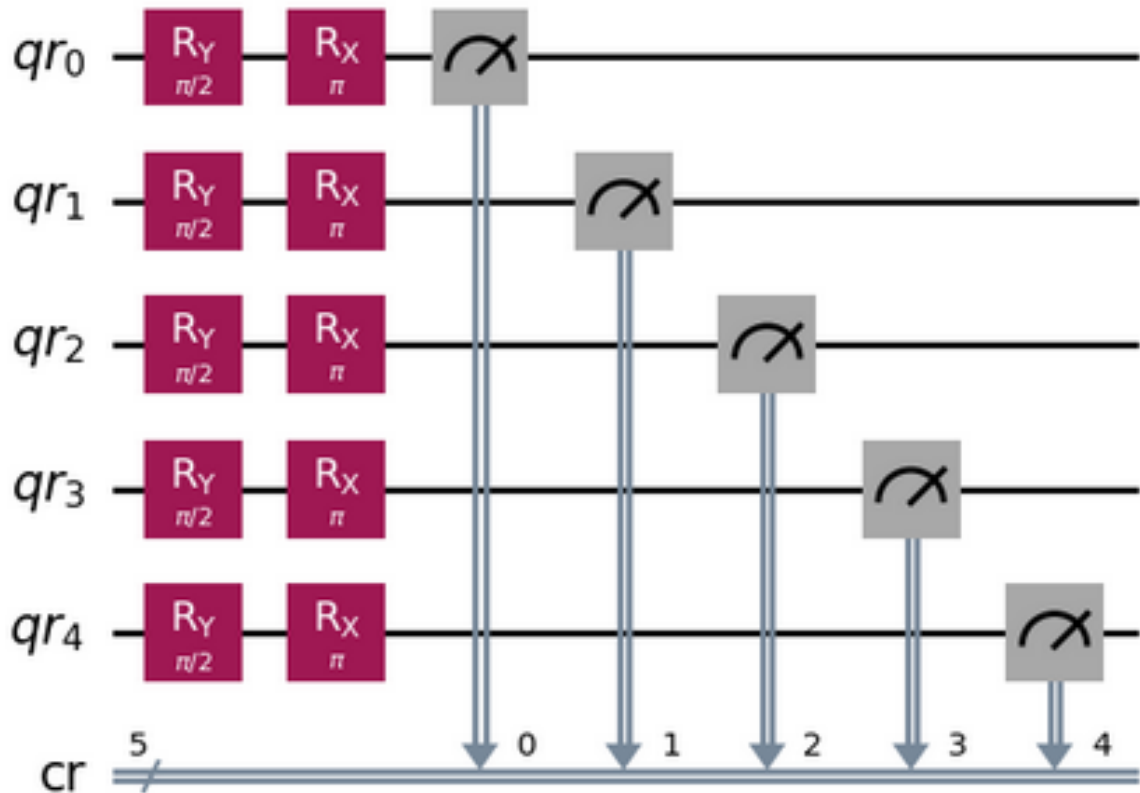
**Figura 4.15:** Circuito random\_binary\_number\_circuit original

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10
qb-0	u2(int,float)	' '	' '	' '	' '	measure()	' '	' '	' '	' '
qb-1	' '	u2(int,float)	' '	' '	' '	' '	measure()	' '	' '	' '
qb-2	' '	' '	u2(int,float)	' '	' '	' '	' '	measure()	' '	' '
qb-3	' '	' '	' '	u2(int,float)	' '	' '	' '	' '	measure()	' '
qb-4	' '	' '	' '	' '	u2(int,float)	' '	' '	' '	' '	measure()
cb-0	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
cb-1	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
cb-2	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
cb-3	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
cb-4	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '

**Figura 4.16:** Matrice random\_binary\_number\_circuit original

Circuito random\_binary\_number\_circuit con transpilazione rpcx:



**Figura 4.17:** Circuito random\_binary\_number\_circuit rpcx

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
qb-0	ry(float)	rx(float)	.	.	.	.	.	.	.	.	measure()	.	.	.	.
qb-1	.	.	ry(float)	rx(float)	.	.	.	.	.	.	.	measure()	.	.	.
qb-2	.	.	.	.	ry(float)	rx(float)	.	.	.	.	.	.	measure()	.	.
qb-3	.	.	.	.	.	.	ry(float)	rx(float)	.	.	.	.	.	measure()	.
qb-4	.	.	.	.	.	.	.	.	ry(float)	rx(float)	.	.	.	.	measure()
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

**Figura 4.18:** Matrice random\_binary\_number\_circuit rpcx

Circuito random\_binary\_number\_circuit con transpilazione simple:

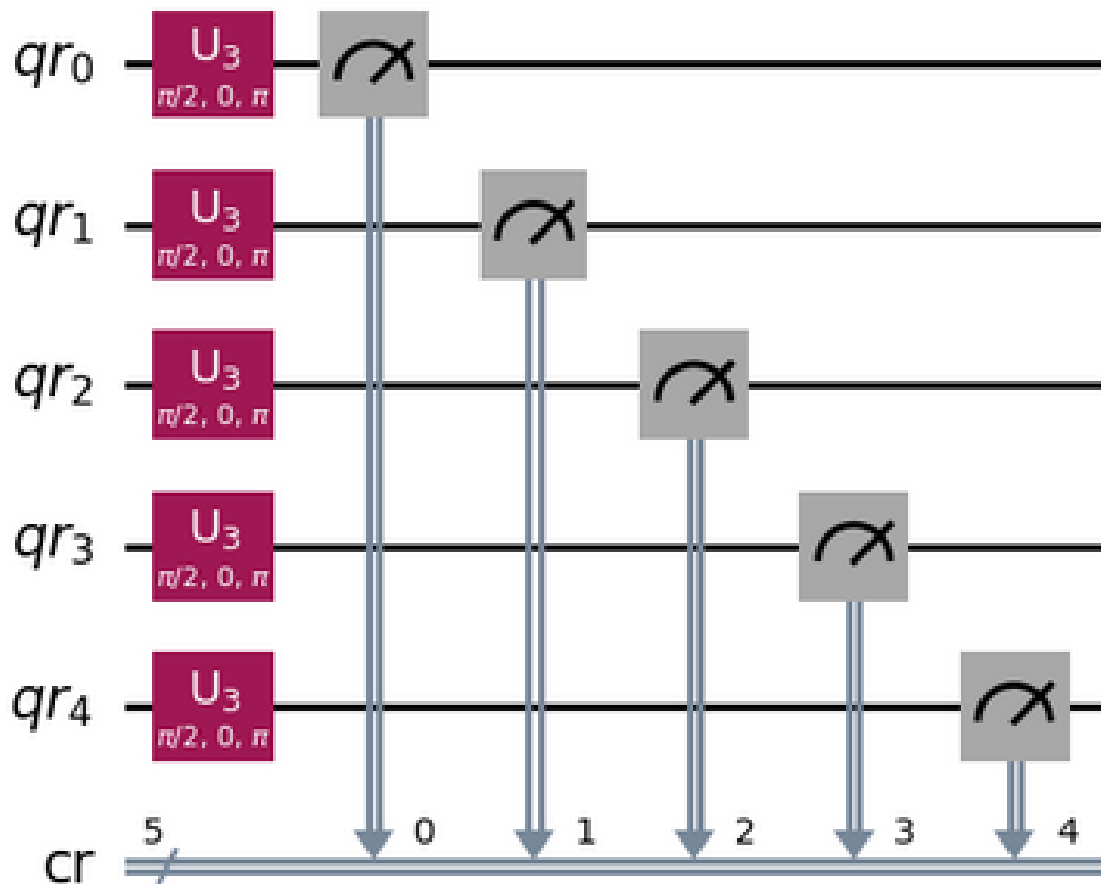


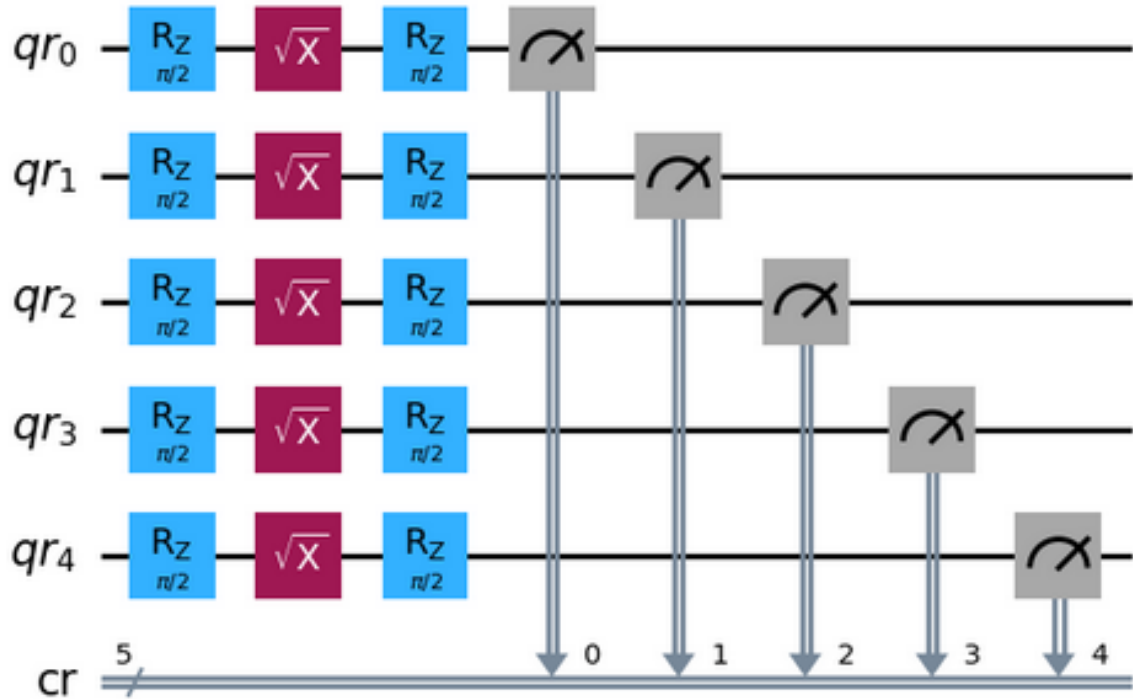
Figura 4.19: Circuito random\_binary\_number\_circuit simple

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10
qb-0	u3(float,int,float)	..	..	..	..	measure()	..	..	..	..
qb-1	..	u3(float,int,float)	..	..	..	..	measure()	..	..	..
qb-2	..	..	u3(float,int,float)	..	..	..	..	measure()	..	..
qb-3	..	..	..	u3(float,int,float)	..	..	..	..	measure()	..
qb-4	..	..	..	..	u3(float,int,float)	..	..	..	..	measure()
cb-0	..	..	..	..	..	..	..	..	..	..
cb-1	..	..	..	..	..	..	..	..	..	..
cb-2	..	..	..	..	..	..	..	..	..	..
cb-3	..	..	..	..	..	..	..	..	..	..
cb-4	..	..	..	..	..	..	..	..	..	..

Figura 4.20: Matrice random\_binary\_number\_circuit simple

Circuito random\_binary\_number\_circuit con transpilazione ibm\_perth:



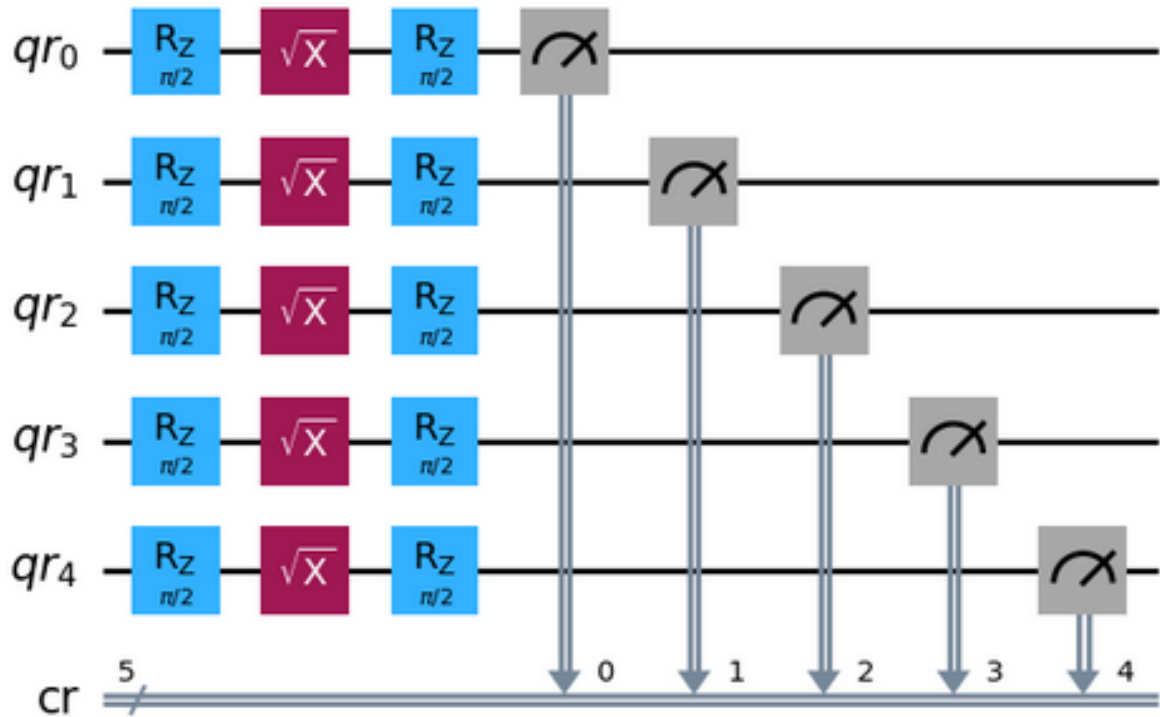
**Figura 4.21:** Circuito random\_binary\_number\_circuit ibm\_perth

Matrice associata:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
qb-0	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.
qb-1	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.
qb-2	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	measure()	.	.
qb-3	.	.	.	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	measure()	.
qb-4	.	.	.	.	.	.	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	measure()
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

**Figura 4.22:** Matrice random\_binary\_number\_circuit ibm\_perth

Circuito random\_binary\_number\_circuit con transpilazione ibm\_sherbroke:



**Figura 4.23:** Circuito random\_binary\_number\_circuit ibm\_sherbroke

Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
qb-0	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.
qb-1	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.
qb-2	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	.	.	measure()	.	.
qb-3	.	.	.	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	.	.	measure()	.
qb-4	.	.	.	.	.	.	.	.	.	.	.	.	rz(float)	sx()	rz(float)	.	.	.	.	measure()
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

**Figura 4.24:** Matrice random\_binary\_number\_circuit ibm\_sherbroke

## 4.2 Sistema AlgoritmiAI

In questo sistema sono presenti i seguenti file .py (in ordine alfabetico):

- ClassicalKMeans.py
- HybridKMeans.py
- SingolaEsecuzioneHybridKMeans.py
- utils.py

Circuito analizzato:

- Circuito `quantum_point_classification_circuit` presente nel file `SingolaEsecuzioneHybridKMeans.py`.

### **File `utils.py`**

Il file `utils.py` è un modulo Python contenente le funzioni utili per gli altri file Python presenti nel sistema.

### **Algoritmo k-means:**

L'algoritmo k-means è una tecnica di clustering utilizzata per suddividere un insieme di dati in k gruppi distinti. L'obiettivo è minimizzare la varianza all'interno di ciascun gruppo. Possiamo dividere l'algoritmo in una sequenza di passi:

1. **Inizializzazione:** Si scelgono k centroidi iniziali (punti centrali dei cluster).
2. **Assegnazione:** Ogni punto dati viene assegnato al centroide più vicino.
3. **Aggiornamento:** Si ricalcolano i centroidi come la media dei punti assegnati a ciascun cluster.
4. **Iterazione:** Si ripetono i passaggi di assegnazione e aggiornamento fino a quando i centroidi non cambiano più significativamente.

L'algoritmo termina quando i centroidi si stabilizzano, indicando che i cluster sono stati definiti.

La versione classica e la versione ibrida cambiano perché nella prima si utilizza un algoritmo classico per trovare il cluster per un punto mentre nella versione ibrida viene utilizzato un algoritmo quantistico per trovare il cluster di un punto mentre, l'algoritmo per individuare i nuovi k centroidi è lo stesso algoritmo classico per entrambe le versioni (perciò abbiamo una versione classica e una versione ibrida).

### Circuito quantum\_point\_classification\_circuit senza transpilazione:

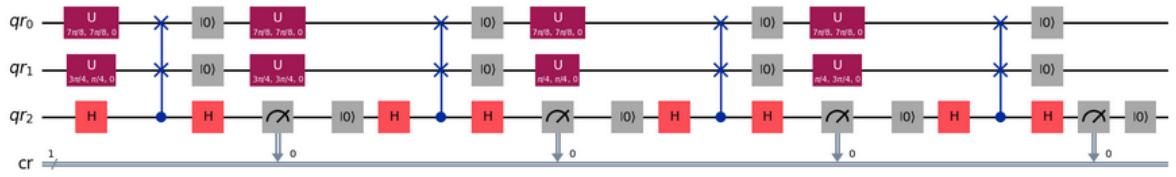


Figura 4.25: Circuito quantum\_point\_classification\_circuit senza transpilazione

### Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
qb-0	u(float,float,int)						reset()					u(float,float,int)				reset()				u(float,float,int)				
qb-1			u(float,float,int)									u(float,float,int)										u(float,float,int)		
qb-2	h()						measure()																	measure()
cr-0																								
qb-0	reset()																							
qb-1																								
qb-2																								
cr-0																								

Figura 4.26: Matrice quantum\_point\_classification\_circuit senza transpilazione

### Circuito quantum\_point\_classification\_circuit con transpilazione original:

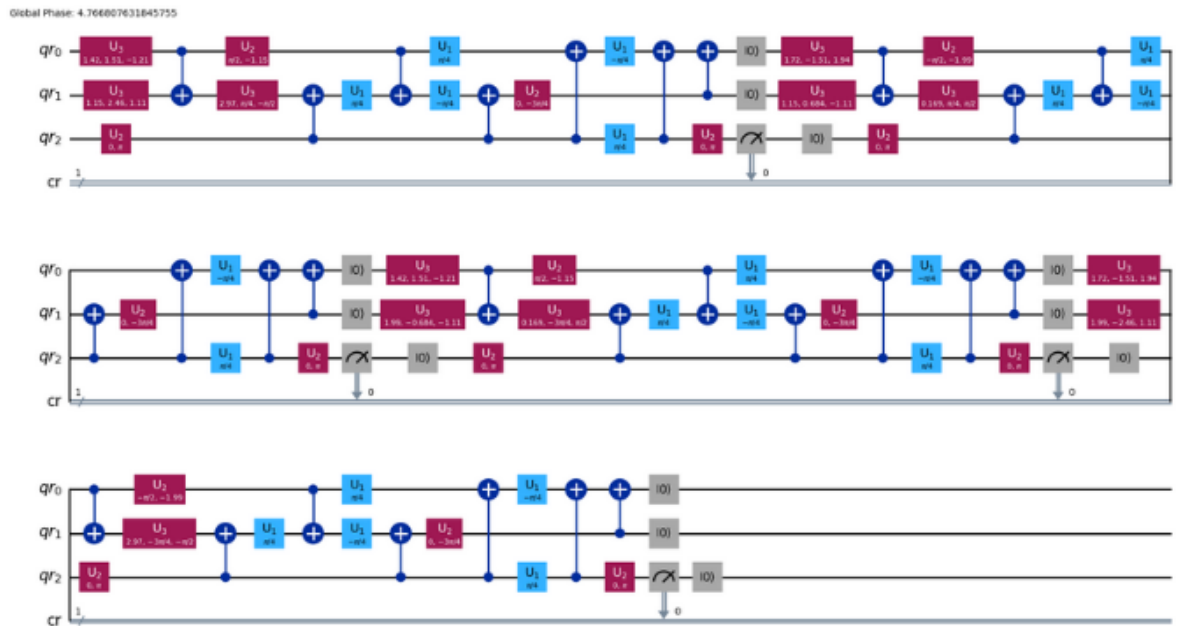


Figura 4.27: Circuito quantum\_point\_classification\_circuit original

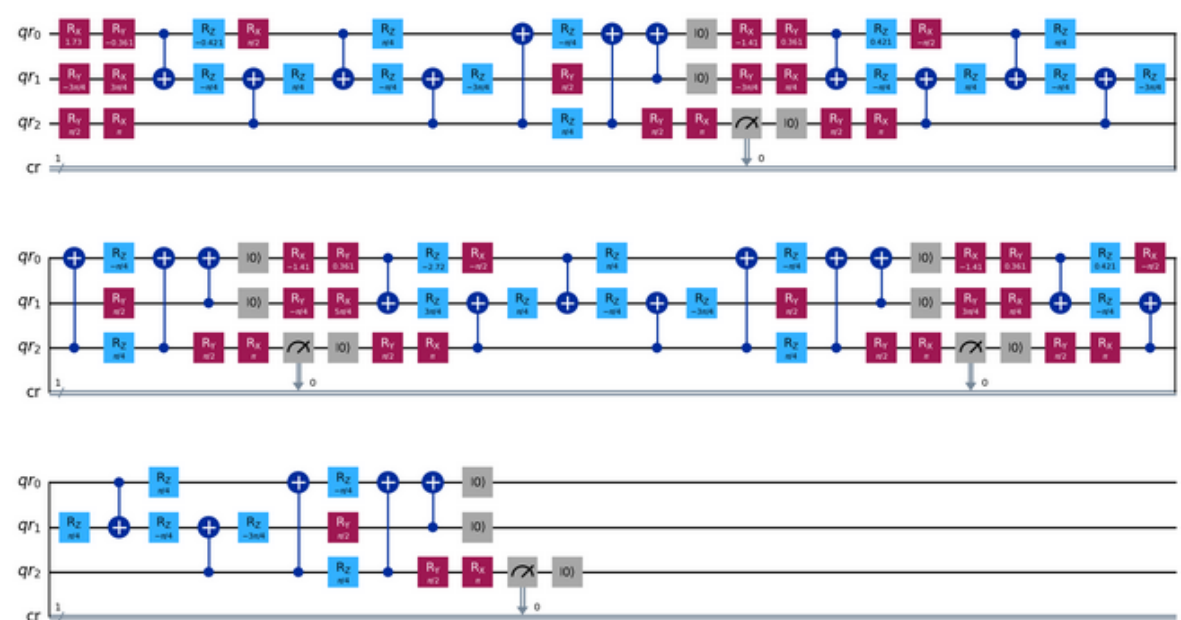


**Matrice associata:**

[illegible]

**Figura 4.28:** Matrice quantum\_point\_classification\_circuit original

Circuito quantum\_point\_classification\_circuit con transpilazione rpcx:



**Figura 4.29:** Circuito quantum\_point\_classification\_circuit rpcx

**Matrice associata:**

[illegible]

**Figura 4.30:** Matrice quantum\_point\_classification\_circuit rpx

### Circuito quantum\_point\_classification\_circuit con transpilazione simple:

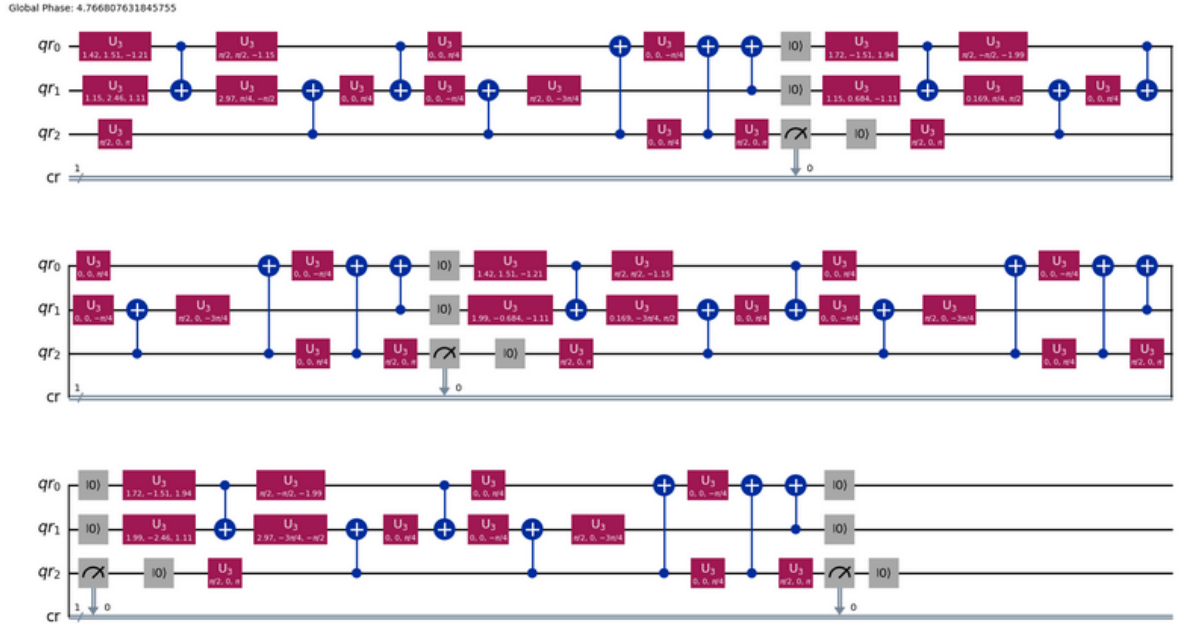


Figura 4.31: Circuito quantum\_point\_classification\_circuit simple

### Matrice associata:

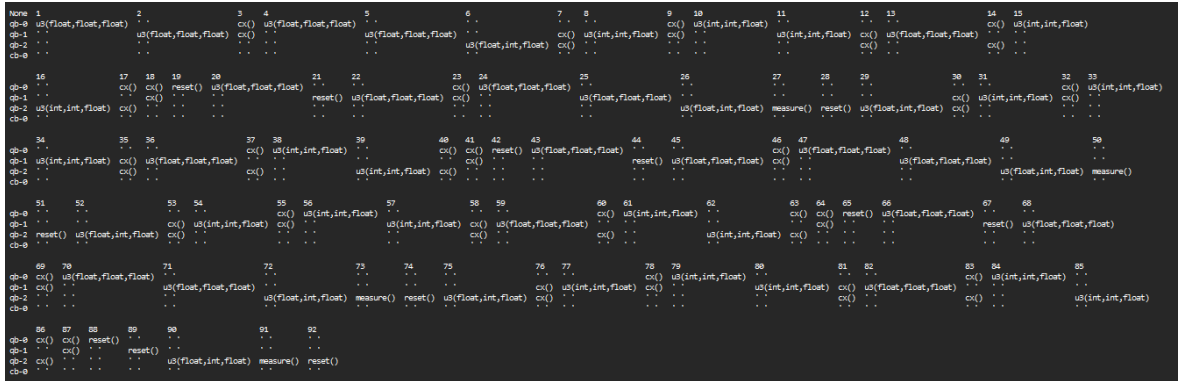


Figura 4.32: Matrice quantum\_point\_classification\_circuit simple

Circuito quantum\_point\_classification\_circuit con transpilazione ibm\_perth:

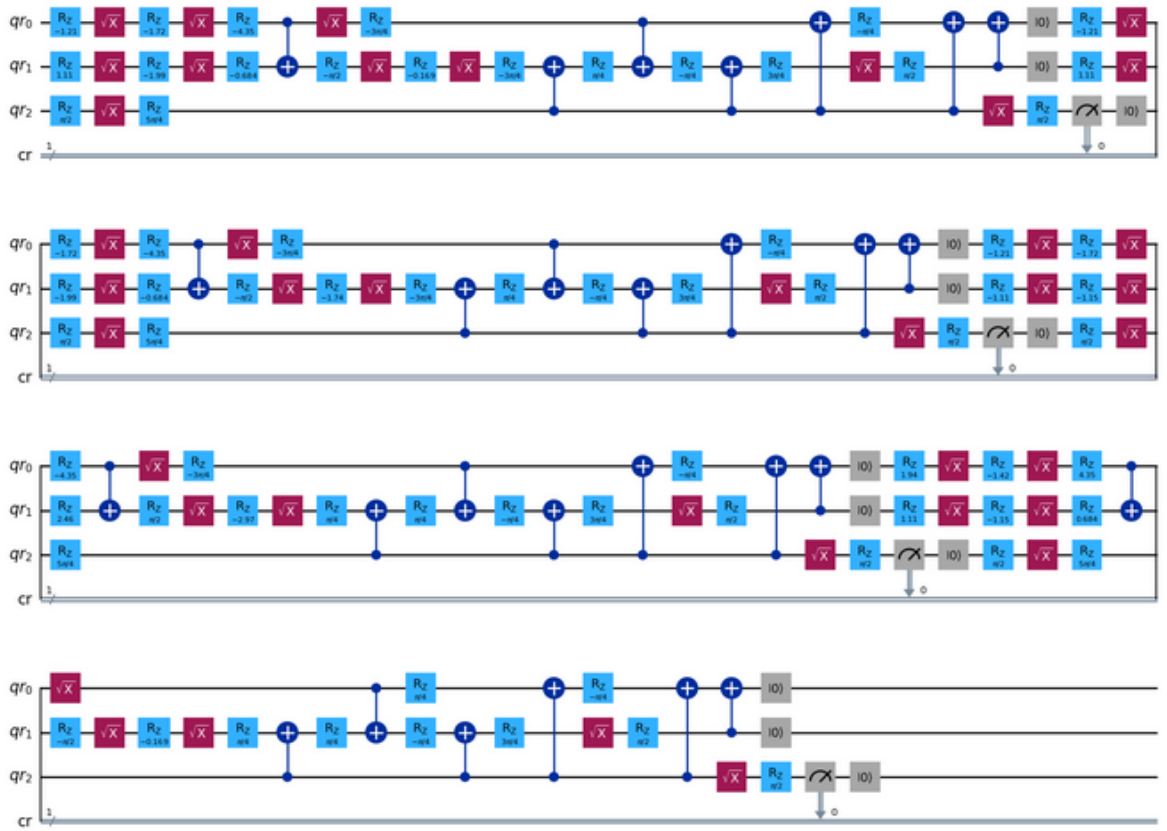


Figura 4.33: Circuito quantum\_point\_classification\_circuit ibm\_perth

Matrice associata:

	None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
qb-0	x()	barrier()	.	.	.	.	cx()	rz(float)	sx()	rz(float)	barrier()	measure()	.	.	barrier()	.	.	.	.	cx()	.	.	.	.
qb-1	.	barrier()	rz(float)	sx()	rz(float)	cx()	cx()	.	.	.	.	barrier()	.	measure()	barrier()	cx()	.	.	.	.	.	.	.	.
qb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	barrier()	cx()	rz(float)	sx()	rz(float)	cx()	rz(float)	sx()	rz(float)
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.34: Matrice quantum\_point\_classification\_circuit ibm\_perth

## Circuito quantum\_point\_classification\_circuit con transpilazione ibm\_sherbrooke:

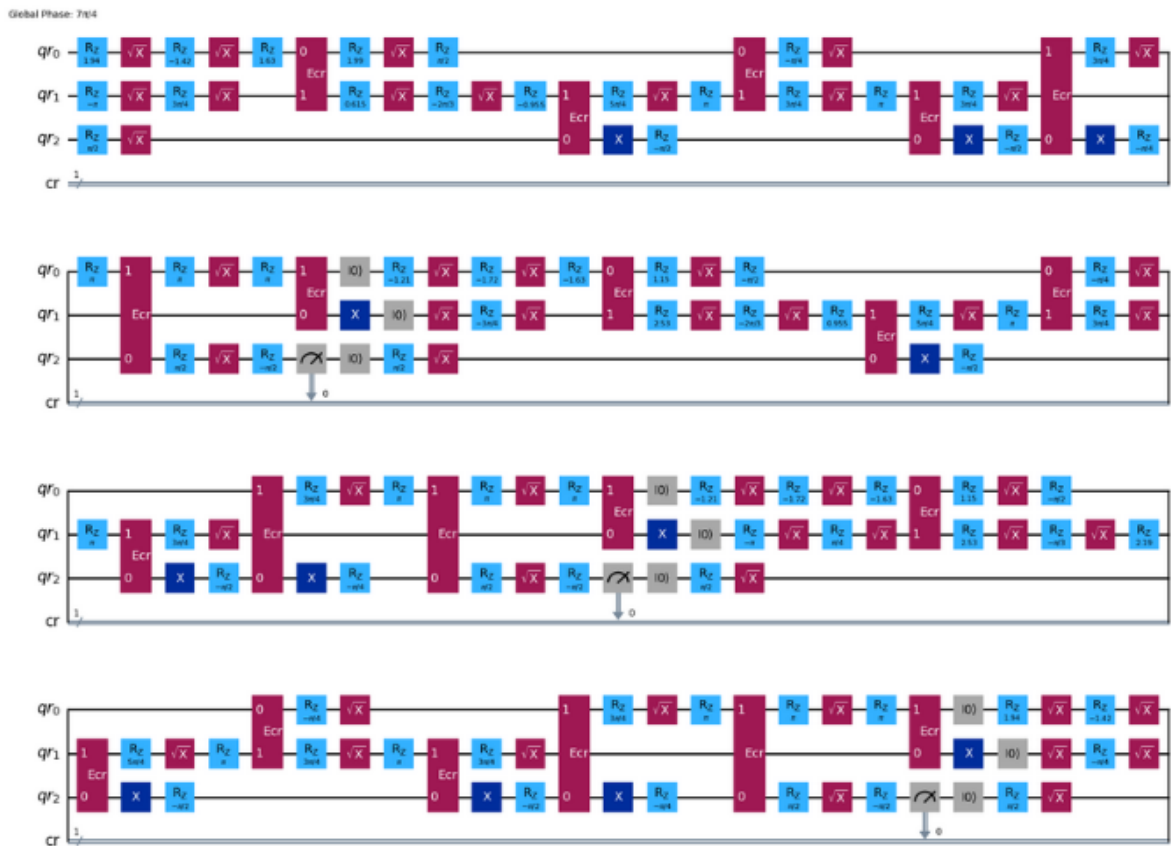


Figura 4.35: Circuito quantum\_point\_classification\_circuit ibm\_sherbrooke parte 1

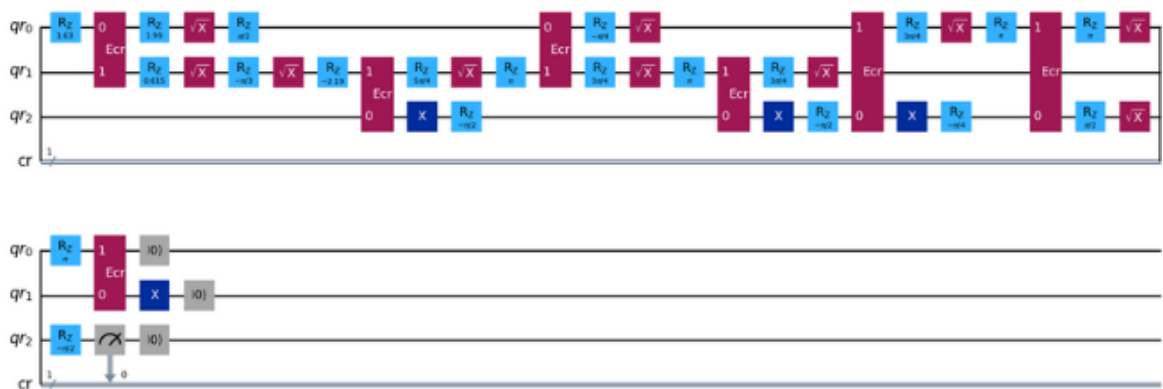


Figura 4.36: Circuito quantum\_point\_classification\_circuit ibm\_sherbrooke parte 2

Matrice associata:

Figura 4.37: Matrice quantum\_point\_classification\_circuit ibm\_sherbroke

## 4.3 Sistema AlgoritmiSicurezza

In questo sistema sono presenti i seguenti file .py (in ordine alfabetico):

- EsecuzioneShor15.py
- FattorizzazioneClassica.py
- RSA.py
- utils.py
- VisualizzazioneCircuitiShor15.py

Circuiti analizzati:

- Circuito a\_mod\_15\_circuit presente nel file VisualizzazioneCircuitiShor15.py.
- Circuito qft\_inverse\_circuit presente nel file VisualizzazioneCircuitiShor15.py.
- Circuito shor\_15\_circuit presente nel file EsecuzioneShor15.py.

### File utils.py

Il file utils.py è un modulo Python contenente le funzioni utili per gli altri file Python presenti nel sistema.

**Fattorizzazione classica e fattorizzazione quantistica:**

La fattorizzazione classica è il processo di scomposizione di un numero intero in un prodotto di numeri primi. Ad esempio, il numero 36 può essere scomposto in  $(2^2 \times 3^2)$ . Questo processo è fondamentale in molti campi della matematica e della crittografia.

L'algoritmo di fattorizzazione classica considerato non è quello attualmente più efficiente in termini di tempo.

L'**algoritmo di Shor**, sviluppato da Peter Shor nel 1994, è un algoritmo quantistico che permette di fattorizzare numeri interi in tempo polinomiale, sfruttando le proprietà della meccanica quantistica. Questo algoritmo è particolarmente importante perché può risolvere problemi di fattorizzazione molto più velocemente rispetto agli algoritmi classici, mettendo a rischio molti sistemi di crittografia attualmente utilizzati come per esempio l'algoritmo RSA che vedremo alla fine. L'algoritmo di Shor utilizza i circuiti `a_mod_x`, `c_a_mod_x` e trasformata di Fourier quantistica Inversa (QFT inversa).

**Circuito `a_mod_x`:**

Il circuito `a_mod_x` è utilizzato nell'algoritmo di Shor per calcolare il resto della divisione di `a` per `x`. Questo circuito è essenziale per determinare l'ordine di un numero modulo `x`, che è un passo cruciale nell'algoritmo di Shor.

**Circuito `c_a_mod_x`:**

Il circuito `c_a_mod_x` è una versione controllata del circuito `a_mod_x`. In questo circuito, un controllo aggiuntivo permette di eseguire l'operazione `a_mod_x` solo quando un qubit di controllo è in uno stato specifico, questo è utile per implementare operazioni condizionali nell'algoritmo di Shor.

**Trasformata di Fourier Quantistica Inversa (QFT inversa):**

La trasformata di Fourier quantistica inversa è una trasformazione lineare su qubit che è l'analogo quantistico della trasformata discreta di Fourier inversa. La QFT inversa è utilizzata nell'algoritmo di Shor per estrarre informazioni dalle fasi relative tra gli stati quantistici. Questo passaggio è cruciale per determinare l'ordine di un numero modulo `x` e, di conseguenza, per la fattorizzazione del numero.

L'algoritmo di Shor considerato è la versione per eseguire la fattorizzazione di 15.

**Circuiti quantistici del file `VisualizzazioneCircuitiShor15.py`:**

Poiché questo file ha ben 2 circuiti allora ho pensato di inserire solamente i circuiti senza transpilazione.

Circuito a\_mod\_15\_circuit senza transpilazione:

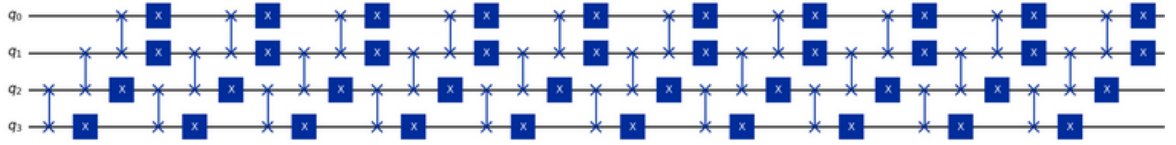


Figura 4.38: Circuito a\_mod\_15\_circuit

Matrice associata:

	None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
qb-0	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
qb-1	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
qb-2	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
qb-3	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..

Figura 4.39: Matrice a\_mod\_15\_circuit

Circuito qft\_inverse\_circuit senza transpilazione:

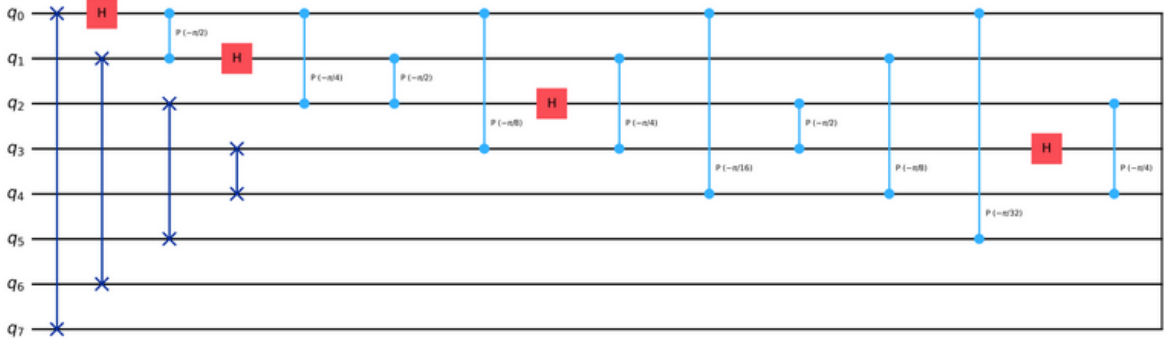


Figura 4.40: Circuito qft\_inverse\_circuit parte 1

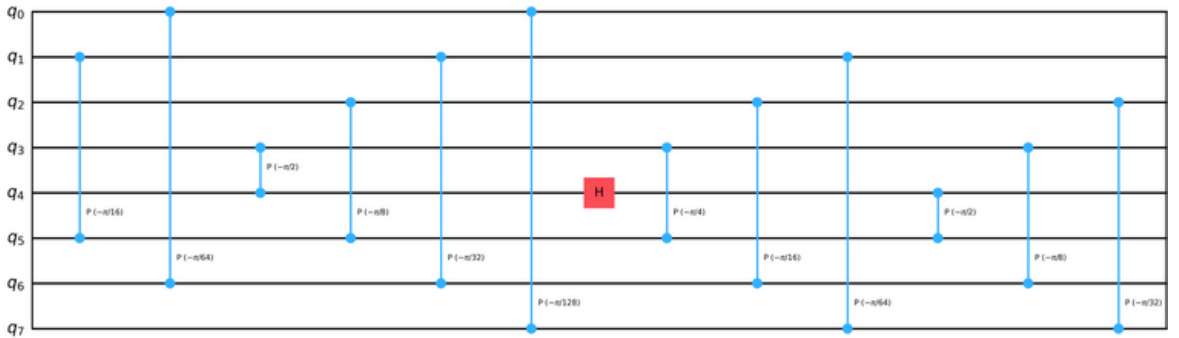


Figura 4.41: Circuito qft\_inverse\_circuit parte 2

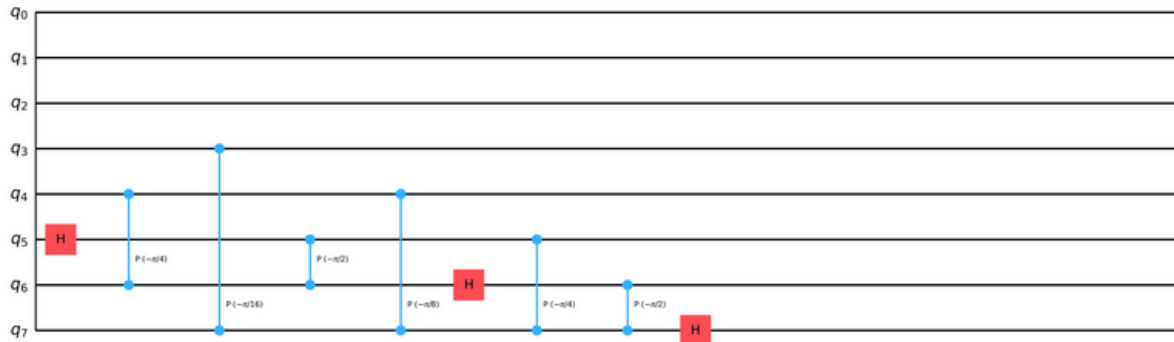


Figura 4.42: Circuito qft\_inverse\_circuit parte 3

Matrice associata:

Row	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
qb-0	swap()	.	.	.	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)
qb-1	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)
qb-2	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.
qb-3	.	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)
qb-4	.	.	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.
qb-5	.	.	.	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)
qb-6	.	.	.	.	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.
qb-7	.	.	.	.	.	.	.	swap()	.	.	.	cp(float)	h()	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)	.	cp(float)

Figura 4.43: Matrice qft\_inverse\_circuit

File EsecuzioneShor15.py:

Per quanto riguarda questo file, verrà eseguita solamente l'analisi senza transpilazione.

Circuito shor\_15\_circuit senza transpilazione:

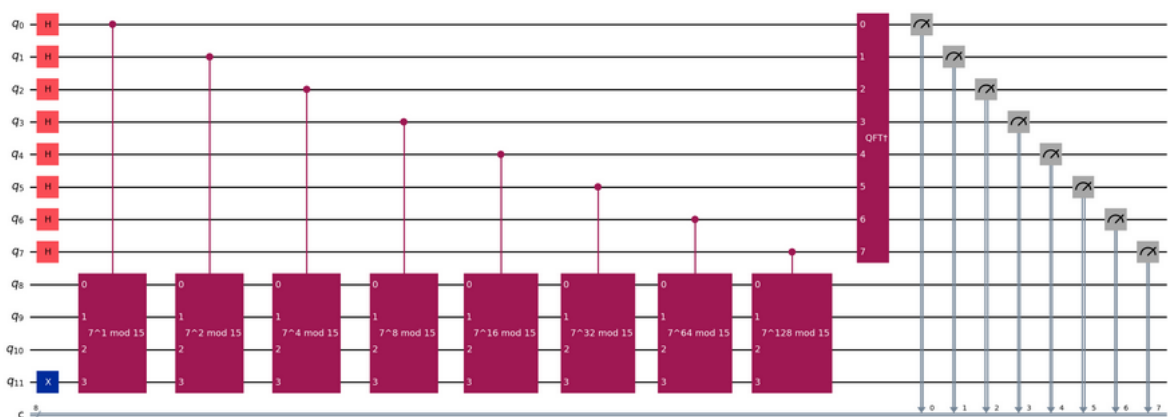


Figura 4.44: Circuito shor\_15\_circuit senza transpilazione



Matrice associata:

None	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
qb-0	h()	.	.	.	.	.	.	.	.	$c7^1 \text{ mod } 15()$	.	.	.	.	.	.	.	QFTT()
qb-1	.	h()	.	.	.	.	.	.	.	.	$c7^2 \text{ mod } 15()$	.	.	.	.	.	.	QFTT()
qb-2	.	.	h()	.	.	.	.	.	.	.	.	$c7^4 \text{ mod } 15()$	.	.	.	.	.	QFTT()
qb-3	.	.	.	h()	.	.	.	.	.	.	.	.	$c7^8 \text{ mod } 15()$	.	.	.	.	QFTT()
qb-4	.	.	.	.	h()	.	.	.	.	.	.	.	.	$c7^{16} \text{ mod } 15()$	.	.	.	QFTT()
qb-5	.	.	.	.	.	h()	.	.	.	.	.	.	.	.	$c7^{32} \text{ mod } 15()$	.	.	QFTT()
qb-6	.	.	.	.	.	.	h()	.	.	.	.	.	.	.	.	$c7^{64} \text{ mod } 15()$	.	QFTT()
qb-7	.	.	.	.	.	.	.	h()	.	.	.	.	.	.	.	.	$c7^{128} \text{ mod } 15()$	QFTT()
qb-8	.	.	.	.	.	.	.	.	$c7^1 \text{ mod } 15()$	$c7^2 \text{ mod } 15()$	$c7^4 \text{ mod } 15()$	$c7^8 \text{ mod } 15()$	$c7^{16} \text{ mod } 15()$	$c7^{32} \text{ mod } 15()$	$c7^{64} \text{ mod } 15()$	$c7^{128} \text{ mod } 15()$	.	.
qb-9	.	.	.	.	.	.	.	.	$c7^1 \text{ mod } 15()$	$c7^2 \text{ mod } 15()$	$c7^4 \text{ mod } 15()$	$c7^8 \text{ mod } 15()$	$c7^{16} \text{ mod } 15()$	$c7^{32} \text{ mod } 15()$	$c7^{64} \text{ mod } 15()$	$c7^{128} \text{ mod } 15()$	.	.
qb-10	.	.	.	.	.	.	.	.	$c7^1 \text{ mod } 15()$	$c7^2 \text{ mod } 15()$	$c7^4 \text{ mod } 15()$	$c7^8 \text{ mod } 15()$	$c7^{16} \text{ mod } 15()$	$c7^{32} \text{ mod } 15()$	$c7^{64} \text{ mod } 15()$	$c7^{128} \text{ mod } 15()$	.	.
qb-11	.	.	.	.	.	.	.	x()	$c7^1 \text{ mod } 15()$	$c7^2 \text{ mod } 15()$	$c7^4 \text{ mod } 15()$	$c7^8 \text{ mod } 15()$	$c7^{16} \text{ mod } 15()$	$c7^{32} \text{ mod } 15()$	$c7^{64} \text{ mod } 15()$	$c7^{128} \text{ mod } 15()$	.	.
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-0	measure()	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-1	.	measure()	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-2	.	.	measure()	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-3	.	.	.	measure()	.	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-4	.	.	.	.	measure()	.	.	.	.	.	.	.	.	.	.	.	.	.
qb-5	.	.	.	.	.	measure()	.	.	.	.	.	.	.	.	.	.	.	.
qb-6	.	.	.	.	.	.	measure()	.	.	.	.	.	.	.	.	.	.	.
qb-7	.	.	.	.	.	.	.	measure()	.	.	.	.	.	.	.	.	.	.
qb-8	.	.	.	.	.	.	.	.	measure()	.	.	.	.	.	.	.	.	.
qb-9	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.	.	.	.	.
qb-10	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.	.	.	.
qb-11	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.	.	.
cb-0	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.	.
cb-1	.	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.	.
cb-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.	.
cb-3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.	.
cb-4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	measure()	.
cb-5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	measure()
cb-6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
cb-7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 4.45: Matrice shor\_15\_circuit senza transpilazione

**Algoritmo RSA:**

L'algoritmo RSA è uno degli algoritmi di crittografia classica più sicuro al momento però, in futuro, quando i computer quantistici saranno più potenti, allora, potremmo attaccarlo tranquillamente tramite l'algoritmo di Shor.

**Come funziona l'algoritmo RSA:**

- L'algoritmo RSA è un metodo di crittografia asimmetrica utilizzato per proteggere le comunicazioni.
- **Generazione delle chiavi:** Si scelgono due numeri primi grandi e si calcola il loro prodotto, che diventa parte della chiave pubblica.
- **Crittografia:** Il mittente utilizza la chiave pubblica del destinatario per crittografare il messaggio.
- **Decrittografia:** Il destinatario utilizza la propria chiave privata per decrittografare il messaggio.

**Come attaccare l'algoritmo RSA:**

- Con l'algoritmo RSA si scelgono 2 numeri primi grandi  $p$  e  $q$  e si ricava  $n = p \cdot q$ . Se  $p$  e  $q$  sono grandi allora per fattorizzare  $n$  ci vorrebbe troppo tempo con la fattorizzazione classica mentre, se  $p$  e  $q$  sono piccoli allora anche  $n$  sarà abbastanza piccolo da poter essere fattorizzato in poco tempo rompendo così la sicurezza dell'algoritmo e riuscendo ad attaccarlo.

## 4.4 Risultati quantum code smells ottenuti

Di seguito saranno riportate delle tabelle contenenti i valori ottenuti per i vari quantum code smells:

Nessuna transpilazione		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File EsecuzioneShor15.py	0	0
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.1:** Analisi statica nessuna transpilazione

Nessuna transpilazione						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	4	4	0	2	0	0
Circuito teleportation_circuit	4	4	2	10	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	3	2	1	60	3	0
Sistema AlgoritmiSicurezza						
Circuito shor_15_circuit	8	8	0	72	0	0
Circuito qft_inverse_circuit	31	31	0	18	0	0
Circuito a_mod_15_circuit	31	31	0	60	9	0

**Tabella 4.2:** Analisi dinamica nessuna transpilazione

Transpilazione original		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.3:** Analisi statica transpilazione original

Transpilazione original						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	4	4	0	2	0	0
Circuito teleportation_circuit	4	4	2	12	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	0	1	1	88	2	0
Sistema AlgoritmiSicurezza						
Circuito qft_inverse_circuit	69	0	0	62	29	0
Circuito a_mod_15_circuit	69	0	0	140	20	0

**Tabella 4.4:** Analisi dinamica transpilazione original

Transpilazione rpcx		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.5:** Analisi statica transpilazione rpcx

Transpilazione rpcx						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	8	0	0	3	0	0
Circuito teleportation_circuit	5	5	2	16	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	11	0	1	104	2	0
Sistema AlgoritmiSicurezza						
Circuito qft_inverse_circuit	75	0	0	64	29	0
Circuito a_mod_15_circuit	75	0	0	140	20	0

**Tabella 4.6:** Analisi dinamica transpilazione rpcx

Transpilazione simple		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.7:** Analisi statica transpilazione simple

Transpilazione simple						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	4	4	0	2	0	0
Circuito teleportation_circuit	4	4	2	12	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	8	1	1	88	2	0
Sistema AlgoritmiSicurezza						
Circuito qft_inverse_circuit	80	0	0	62	29	0
Circuito a_mod_15_circuit	80	0	0	140	20	0

**Tabella 4.8:** Analisi dinamica transpilazione simple

Transpilazione ibm_perth		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.9:** Analisi statica transpilazione ibm\_perth

Transpilazione ibm_perth						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	12	0	0	4	0	0
Circuito teleportation_circuit	6	6	2	20	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	20	0	1	168	8	0
Sistema AlgoritmiSicurezza						
Circuito qft_inverse_circuit	76	0	0	66	29	0
Circuito a_mod_15	76	0	0	140	20	0

**Tabella 4.10:** Analisi dinamica transpilazione ibm\_perth

Transpilazione ibm_sherbroke		
Analisi statica		
Sistema VariAlgoritmi		
	LPQ	NC
File NumeroCasuale.py	0	0
File Teletrasporto1Qubit.py	0	0
Sistema AlgoritmiAI		
File SingolaEsecuzioneHybridKMeans.py	0	0
Sistema AlgoritmiSicurezza		
File VisualizzazioneCircuitiShor15.py	0	0

**Tabella 4.11:** Analisi statica transpilazione ibm\_sherbroke

Transpilazione ibm_sherbroke						
Analisi dinamica						
Sistema VariAlgoritmi						
	IdQ	IQ	IM	LC	ROC	CG
Circuito random_binary_number_circuit	12	0	0	4	0	0
Circuito teleportation_circuit	9	0	2	30	0	0
Sistema AlgoritmiAI						
Circuito quantum_point_classification_circuit	25	0	1	186	14	0
Sistema AlgoritmiSicurezza						
Circuito qft_inverse_circuit	253	0	0	154	31	0
Circuito a_mod_15_circuit	253	0	0	606	42	0

**Tabella 4.12:** Analisi dinamica transpilazione ibm\_sherbroke



#### 5.1 Sviluppi futuri

La web application SearchQS, come abbiamo detto nei capitoli precedenti, è stata progettata per analizzare dei sistemi quantistici inseriti in input per poter individuare gli 8 quantum code smells definiti nel capitolo 2.

La web application sviluppata però potrebbe essere migliorata considerando i seguenti compiti:

- **Miglioramento del codice:** si potrebbe riprendere il codice della web application cercando di migliorarlo.
- **Aggiunta di nuovi code smells da analizzare e/o dei nuovi tipi di analisi da effettuare:** come riportato nel capitolo 2, in SearchQS abbiamo considerato solamente i seguenti 8 quantum code smells:
  - CG (use of Customized Gates).
  - ROC (Repeated set of Operations of Circuit).
  - NC (Non-parametrized Circuit).
  - LC (Long Circuit).
  - IM (Intermediate Measurement).

- IdQ (Idle Qubit).
- IQ (Initialization of Qubit).
- LPQ (no-alignment between the Logical and Physical Qubits)

e abbiamo considerato solamente 2 tipi di analisi:

- Analisi statica.
- Analisi dinamica.

Si potrebbe quindi considerare di aggiungere nuovi tipi di quantum code smells da analizzare oltre agli 8 riportati sopra e, perché no, potremmo considerare anche alcuni code smells classici da individuare come per esempio:

- Classe troppo grande (nella programmazione Object Oriented).
- Lista di parametri troppo lunga (per metodi o funzioni).

Inoltre, potremmo considerare nuovi tipi di analisi da effettuare oltre all'analisi statica e all'analisi dinamica.

- **Analisi di codice quantistico scritto in un altro linguaggio di programmazione:** potremmo considerare, oltre al Qiskit offerto dall'azienda IBM, un altro tipo di linguaggio di programmazione quantistico come per esempio Q# offerto da Microsoft. Questo comporterebbe anche l'introduzione di nuovi tipi di transpilazioni che, a differenza dei 5 attualmente offerti da SearchQS per le macchine IBM, ne introdurremmo dei nuovi per le macchine Microsoft.
- **Aggiunta di nuovi tipi di transpilazioni considerate:** le transpilazioni attualmente offerte da SearchQS sono riportate nel capitolo 2 e sono le seguenti:
  - original.
  - ibm\_perth.
  - ibm\_sherbrooke.
  - rpcx.
  - simple.

Potremmo quindi considerare nuovi tipi di transpilazioni. Questo potrebbe essere necessario se considerassimo il punto precedente ovvero, l'analisi di codice quantistico scritto in un altro linguaggio di programmazione.

- **Inserire un modulo per eliminare i code smells individuati:** si potrebbe aggiungere un modulo che ha il compito di prendere in input una matrice analizzata oppure il file analizzato con i code smells individuati e gli algoritmi presenti in questo modulo avranno il compito di eliminare i code smells individuati (il valore del code smell deve diventare 0) oppure almeno abbassare il valore con un nuovo valore più basso. Questo potremmo farlo per esempio considerando le seguenti strategie:
  - Considerando un algoritmo genetico che prende in input una matrice, esegue degli algoritmi che si fondano sui principi genetici dell'evoluzione delle specie e ritorna come output una matrice migliorata e poi da quest'ultima ricavare in qualche modo il circuito quantistico migliore.
  - Considerando un algoritmo di machine learning che prende in considerazione un dataset popolato da vari codici quantistici e cercare in qualche modo di ritornare un codice migliore di quello fornito in input.

## 5.2 Conclusioni

Grazie alla scrittura di questa tesi, siamo riusciti a capire le basi dell'informatica quantistica. L'obiettivo della tesi era appunto l'acquisizione dei concetti base dell'informatica quantistica e dell'esecuzione della web application SearchQS sviluppata durante il tirocinio interno che si occupa dell'analizzare un programma o sistema quantistico per poter individuare i quantum code smells presenti in essa.

Sono partito dalla lettura del documento offerto dal relatore prof. Fabio Palomba che parlava degli 8 quantum code smells da individuare, delle 5 transpilazioni offerte dal sistema SearchQS e del funzionamento dell'analisi statica e dinamica, successivamente ho preso spunto su come poter individuare i quantum code smells tramite il codice sorgente del sistema QSmell presente su github e successivamente ho iniziato a progettare ed implementare la web application SearchQS.

Una volta che la progettazione e l'implementazione è stata completata ho iniziato a scrivere la tesi. Sono partito dallo scrivere una breve bozza del capitolo 1 intitolato: "Introduzione" che riportava una piccola spiegazione abbozzata su tutta la tesi, poi ho individuato i concetti riportati nel capitolo 2 tramite varie fonti prese da internet, dal sito web github e dalla mia conoscenza pregressa. Questo mi ha permesso scrivere il

capitolo 2 intitolato: "Background e stato dell'arte" in cui si parla dei concetti individuati e della differenza tra i sistemi QSmell e SearchQS.

Successivamente, sono passato alla scrittura del capitolo 3 intitolato: "SearchQS: Una Piattaforma Web per l'Individuazione ed Esplorazione di Problemi di Progettazione in Circuiti Quantistici" in cui vengono riportate delle domande di ricerca che mi sono posto e in cui ho dato una risposta e poi viene riportata una breve spiegazione dei metodi e delle funzioni più importanti riguardo l'analisi di un sistema quantistico e su come vengono individuati i vari quantum code smells. Successivamente, ho individuato dei possibili algoritmi quantistici da analizzare e le loro controparti classiche tramite varie fonti, tra cui, internet e github. Ho quindi preso questi algoritmi, li ho modificati per farli funzionare con la versione di Qiskit utilizzata e per i nostri scopi, li ho provati con Jupyter Notebook (file di tipo .ipynb) li ho riscritti come file di tipo Python (file di tipo .py) per poterli analizzare e li ho divisi in 3 sistemi quantistici chiamati: "VariAlgoritmi", "AlgoritmiAI" e "AlgoritmiSicurezza". Quindi, nel capitolo 4 intitolato: "Analisi di alcuni sistemi quantistici" viene riportata una breve spiegazione dei vari algoritmi presenti nei vari sistemi quantistici analizzati e, nella sezione 4.4 vengono riportate delle tabelle, 2 per ogni tipo di analisi effettuata di cui 1 per l'analisi statica ed 1 per l'analisi dinamica. Successivamente, ho scritto il capitolo attuale (il capitolo 5) intitolato "Conclusioni" in cui ho riportato dei possibili sviluppi futuri per il sistema SearchQS e le conclusioni in cui spiego appunto cosa ho fatto in questa tesi e, come ultima cosa, dopo aver completato i capitoli 2, 3, 4 e 5, ho completato il capitolo 1 in cui spiego: il contesto applicativo della tesi, le motivazioni e gli obiettivi della tesi, i risultati ottenuti con la tesi e la struttura della tesi completa.

---

## Bibliografia

---

- [1] Manuel De Stefano, Dario Di Nucci, Fabio Palomba, Andrea De Lucia - "An Empirical Study Into the Effects of Transpilation on Quantum Circuit Smells"
- [2] Sistema QSmell
- [3] Piattaforma web SearchQS e sistemi quantistici analizzati
- [4] Documentazione Qiskit SDK v.1.2.0 (latest)
- [5] Veronica Cristiano, Francesco Stocco - "Il Qubit: introduzione agli algoritmi quantistici"
- [6] Gli spazi di Hilbert
- [7] Lorenzo Pareschi - "Vettori e matrici"
- [8] DETERMINANTE DI UNA MATRICE
- [9] Matrice inversa
- [10] Bra-Ket Notation
- [11] Roberto Campagnola - "Circuiti quantistici"
- [12] Operazioni su più qubit
- [13] Sovrapposizione quantistica abbracciare la doppia natura di Q
- [14] Una spiegazione semplice dell'entanglement quantistico
- [15] Interferenza Quantistica
- [16] Generazione di un numero casuale
- [17] Qiskit Quantum Teleportation (No-Cloning Theorem)
- [18] Clustering lezione corso FIA UniSA
- [19] Algoritmo di Shor per fattorizzare il numero 15

- [20] Algoritmo RSA
- [21] Fattorizzazione classica
- [22] Il futuro dell'informatica l'impatto duraturo della legge di Moore e oltre
- [23] Quantum Computing: storia, sviluppo e applicazioni
- [24] Che cos'è il moto browniano?

---

## Ringraziamenti

---

Dedico questa parte finale della tesi per porre i miei ringraziamenti alle seguenti persone:

Prima di tutto, vorrei ringraziare il mio relatore prof. Fabio Palomba per il suo supporto e la sua guida aiutandomi sia durante il tirocinio interno e sia durante la stesura della tesi.

Ringrazio i miei genitori per il sostegno datomi, per non aver mai mollato insieme a me e per avermi supportato e sopportato.

Ringrazio i miei fratelli, le mie cognate e mio cugino Mario per avermi dato una mano ogni tanto e per avermi supportato e sopportato.

Infine, ringrazio me che non ho mai mollato anche quando sembrava impossibile arrivare a questo traguardo.

Grazie ancora a tutti!!

*Questa tesi ha contribuito a piantare un albero in Ecuador tramite il progetto Treedom.*

<https://www.treedom.net/it/user/sesalab/event/sesa-random-forest>