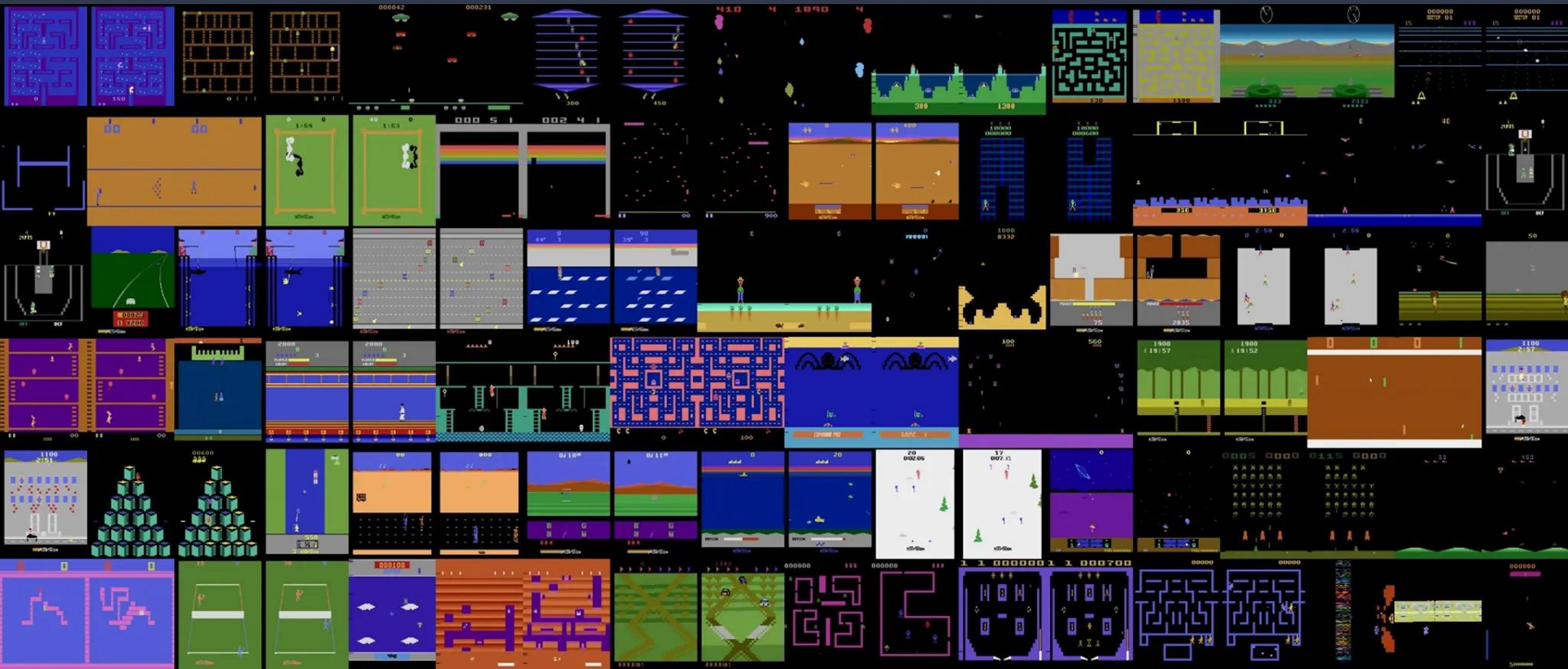
Quantum Enhanced Reinforcement Learning

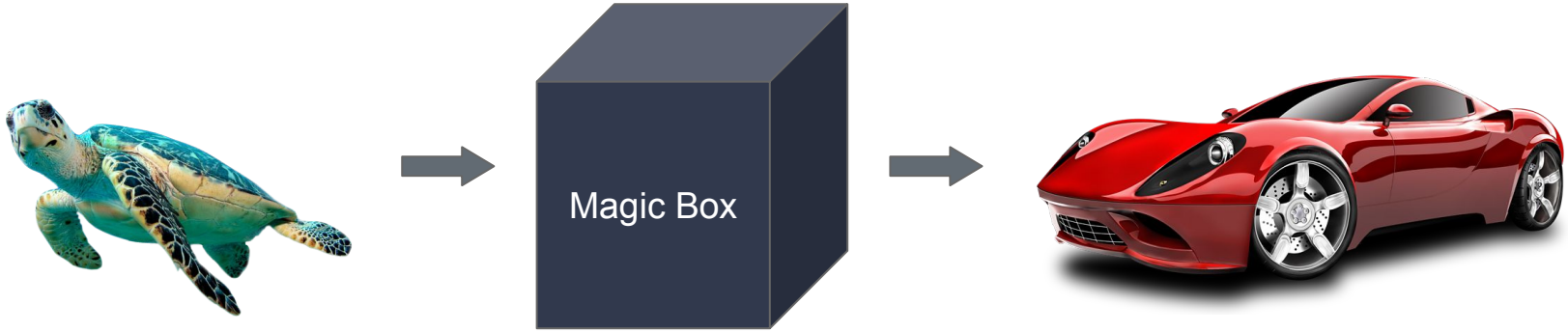
Quantum **M**achine **L**earning course **S**eminar

Speaker: **S**imone **C**aldarella

Motivation



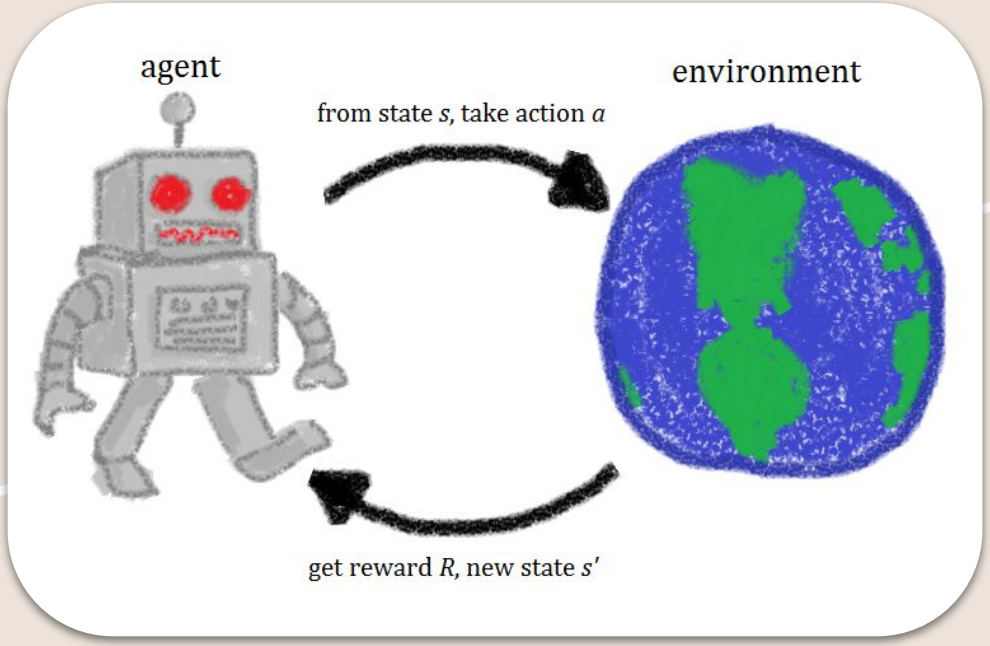
Motivation



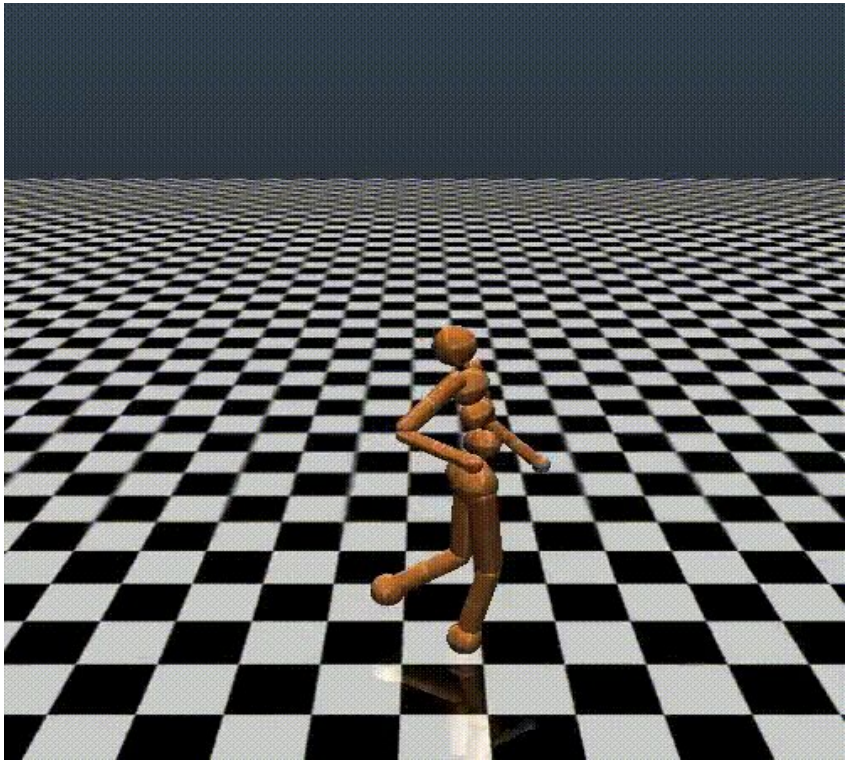
Agenda

1. Reinforcement Learning
   - Introduction
   - Markov Decision Process
   - Value Function
   - Example
2. Quantum Computing
   - Introduction
   - Qubit and Superposition
   - Quantum Circuit
   - Grover Algorithm
3. Quantum Enhanced Reinforcement Learning
   - Setting
   - Implementation
   - Results

Reinforcement Learning



Introduction



When is it used?

- No explicit solution can be found
- No absolute better action can be taken
- **Discover** is required to be a part of the learning
- **Expensive labels**, cheap rewards

Markov Decision Process

A *Markov Decision Process* is a 4-dim tuple $(R,\, S,\, P_a,\, R_a)$:

- $S \rightarrow$ a finite state space
- $A \rightarrow$ a finite actions space
- $P_a\big(s,\, s'\big) \,=\, P\big(\, s_{t+1} \,=\, s' \,\big|\, s_t \,=\, s \,\wedge\, a_t = a\big) \rightarrow$ transition probabilities
- $R_a\big(s,\, s'\big) \rightarrow$ instant reward

The main goal is to find an optimal policy $\pi : s \rightarrow a$ in the given environment, by maximizing the expected future rewards in a potentially infinite process:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t,\, s_{t+1})\right]$$

Value Function

$$V^{\pi}(s_t) = \max_{a_t} \left(R(a_t, s_t) + \gamma \underbrace{\sum_{s_{t+1}} P(s_t, a_t, s_{t+1}) V(s_{t+1})}_{Expected\ value\ of\ future\ state} \right)$$

$$= \mathbf{E}_{\pi} [R_t \mid s_t]$$

$$= \mathbf{E}_{\pi} \left[r_{t+1} + \gamma\, V^{\pi}(s_{t+1}) \mid s_t \right]$$

Value Function - Iteration

Value Iteration Algorithm

Data: θ: a small number
Result: π: a deterministic policy s.t. $\pi \approx \pi_*$
Function *ValueIteration* **is**
    `/* Initialization */`
    Initialize $V(s)$ arbitrarily, except $V(terminal)$;
    $V(terminal) \leftarrow 0$;
    `/* Loop until convergence */`
    $\Delta \leftarrow 0$;
    while $\Delta < \theta$ **do**
        for *each* $s \in S$ **do**
            $v \leftarrow V(s)$;
            $V(s) \leftarrow \max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$;
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
        end
    end
    `/* Return optimal policy */`
    return π s.t. $\pi(s) = arg\max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$;
end

Example

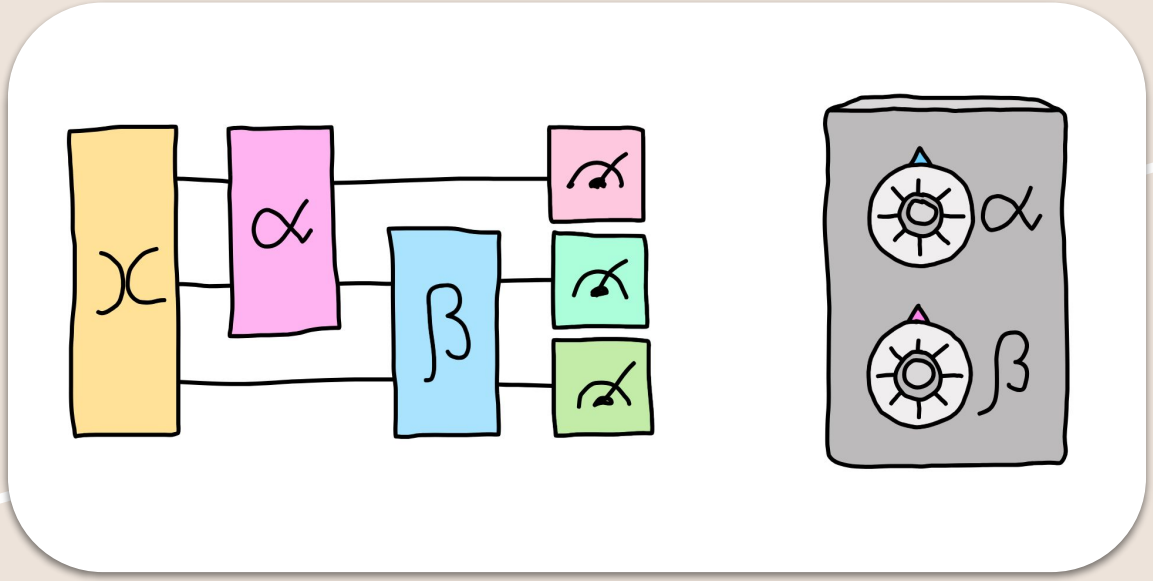
Game Board:

Current state (*s*): 0 0 0 / 0 1 0

Q Table: γ = 0.95

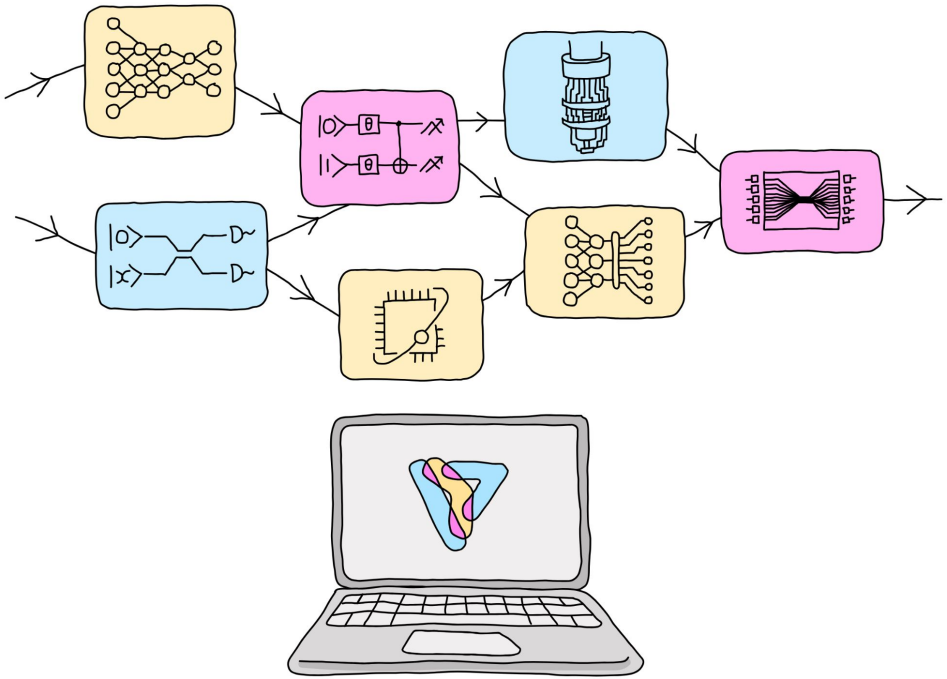
	0 0 0 / 1 0 0	0 0 0 / 0 1 0	0 0 0 / 0 0 1	1 0 0 / 0 0 0	0 1 0 / 0 0 0	0 0 1 / 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Quantum Computing



Introduction

- **Quantum computing** expands classical computers' capabilities by harnessing quantum properties in the physical world
- Leveraging quantum properties to **speed-up** classical algorithms
- Main challenges are **scalability** and **fault tolerance**



Qubits and Superposition

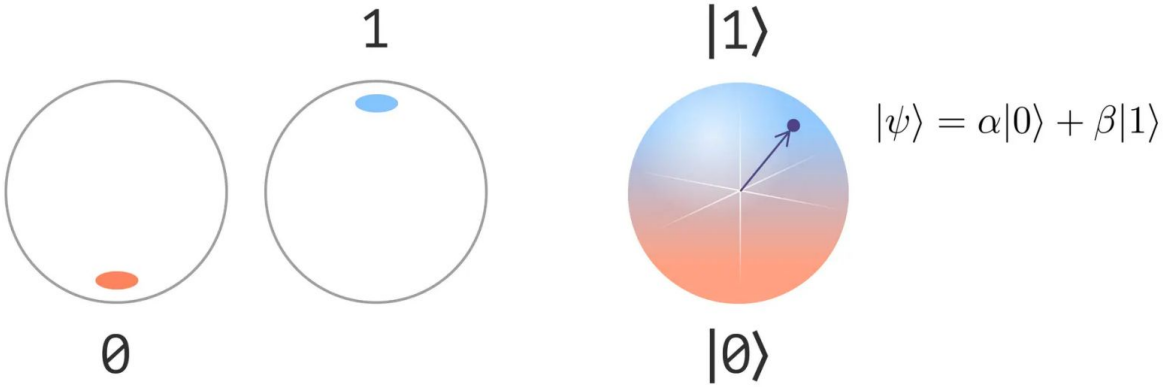
Qubit vs Cbit

- Bit (Cbit) is just a discrete binary representation
- Given N bit we can represent 2^N different states
- Qubit can embed the same number of states, however thanks to the superposition of states, all of these states exists at the same time and can be processed together

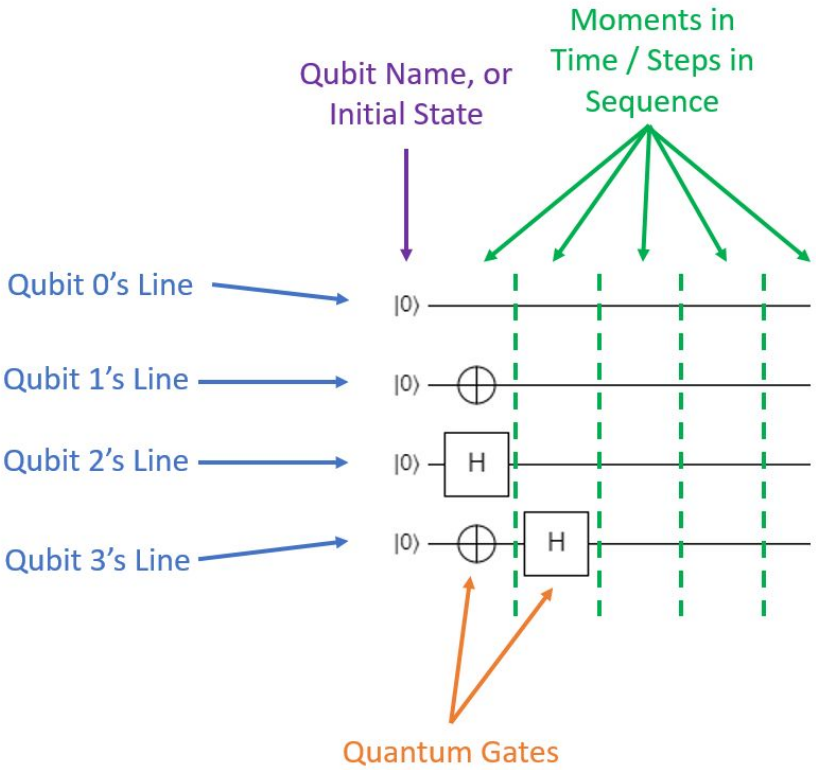
Superposition

- Fundamental property of quantum mechanics for which multiple states of a particle exists together with a degree of probability
- Superposition property derives by the double nature of small “objects” that behave both as waves and particles
- Quantum operations acting on more states happen only in the unobserved theoretical Hilbert Space → when the state is observed it collapse into a single state behaving as a non quantum particle

Qubits and Superposition



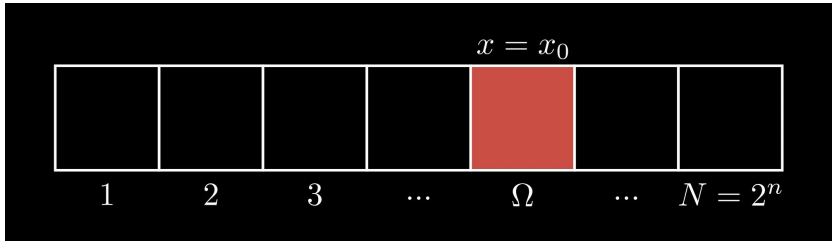
Quantum Circuit



Operator	Gate(s)	Matrix
Pauli-X (X)	X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	H	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	T	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)	Z	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Grover Algorithm - Problem Statement

Given an unsorted database of N items, how can we efficiently find a specific item with only a few queries to the database?



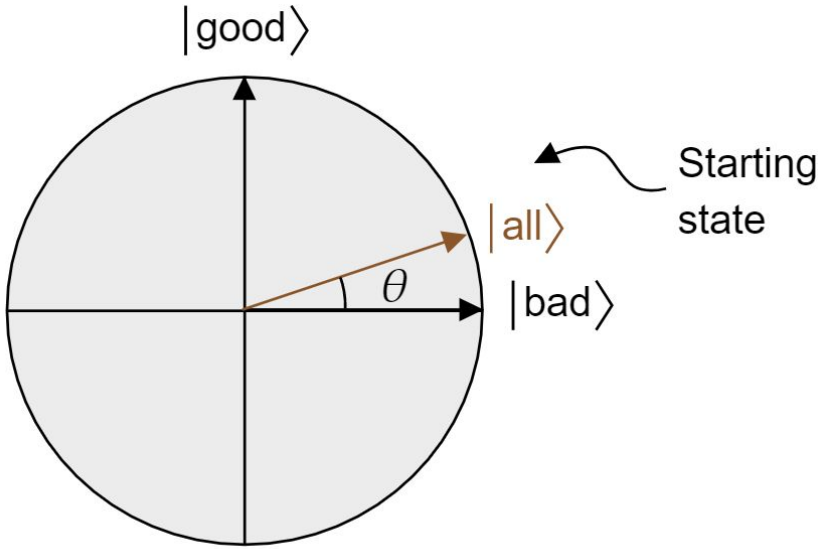
$$\textbf{Grover}(\{0,1\}^n) = \{x_0 | f(x_0) = 1\}$$

$$with\ f(x) : \{0,1\}^n \rightarrow \{0,1\}$$

Grover Algorithm - Problem Statement

1. it is given have an arbitrary state vector in the 2-dim Hilbert space spanned by two orthonormal vectors $\rightarrow$ $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and we want to maximize the probability of observing the state $|1\rangle$, i.e. $\mathcal{P}(|1\rangle) = |\langle\psi|1\rangle|^2 = |\beta|^2$
   - $|0\rangle$ and $|1\rangle$ are just semplifications/proxies that can imply a superposition of more state each one

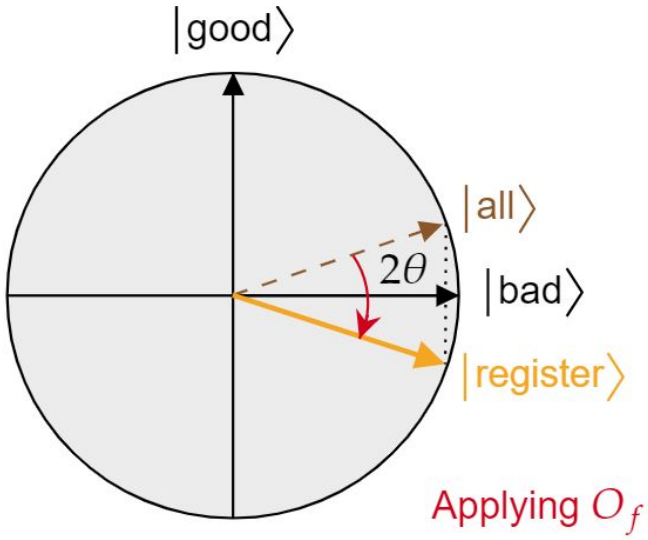
Grover Algorithm - Problem Statement



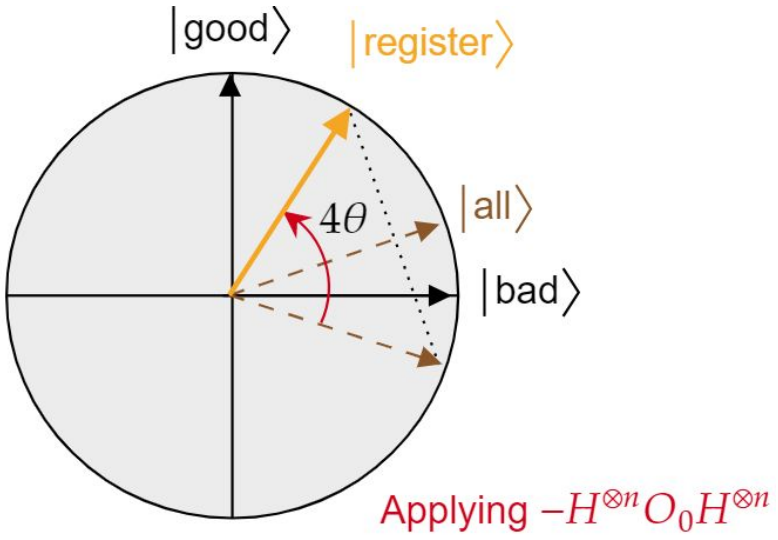
Grover Algorithm - Grover Iteration

2. the state vector $|\psi\rangle$ is then forwarded to an *Oracle* which perform a phase flip of the state to find (1 in this example) $\rightarrow$ $Oracle(|\psi_f\rangle) = \alpha|0\rangle - \beta|1\rangle$
   - the use of an *Oracle* is somewhat controversial in terms of quantum speed-up since theoretical implementations of Grover's Algorithms don't take in account *Oracle*'s complexity
   - the phase flip, which is performed through a Unitary Operator usually implemented through a controlled-Z gate, can be seen as a reflection over the $|0\rangle$ state, i.e. the set of components that are orthogonal to the searched one
3. the so called *diffusion*, is the heart of the Grover's algorithm. It consists on a *reflection* of $|\psi_f\rangle$ w.r.t. $|\psi\rangle$, which is accomplish through the reflection operator $\rightarrow$ $R_{|\psi\rangle} = 2|\psi\rangle\langle\psi| - \mathcal{I}$ applied to $|\psi_f\rangle$, leading to $|\psi_{fr}$

Grover Algorithm - Grover Iteration



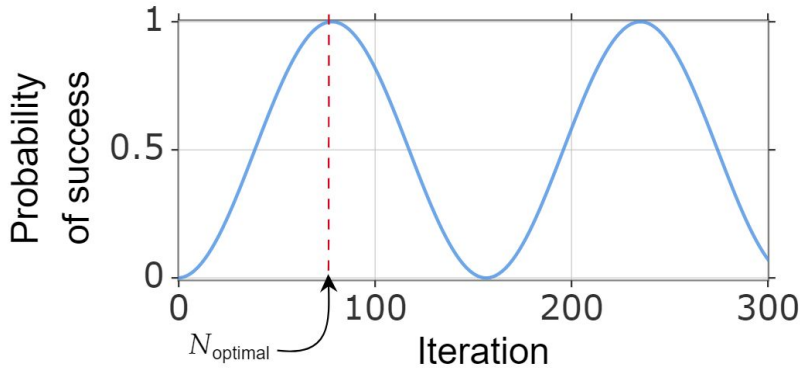
2. Oracle



3. Diffusion

Grover Algorithm – Non Convergence

Since the probability of winning state evolve as a sin^2 it doesn't converge to an optimal solution

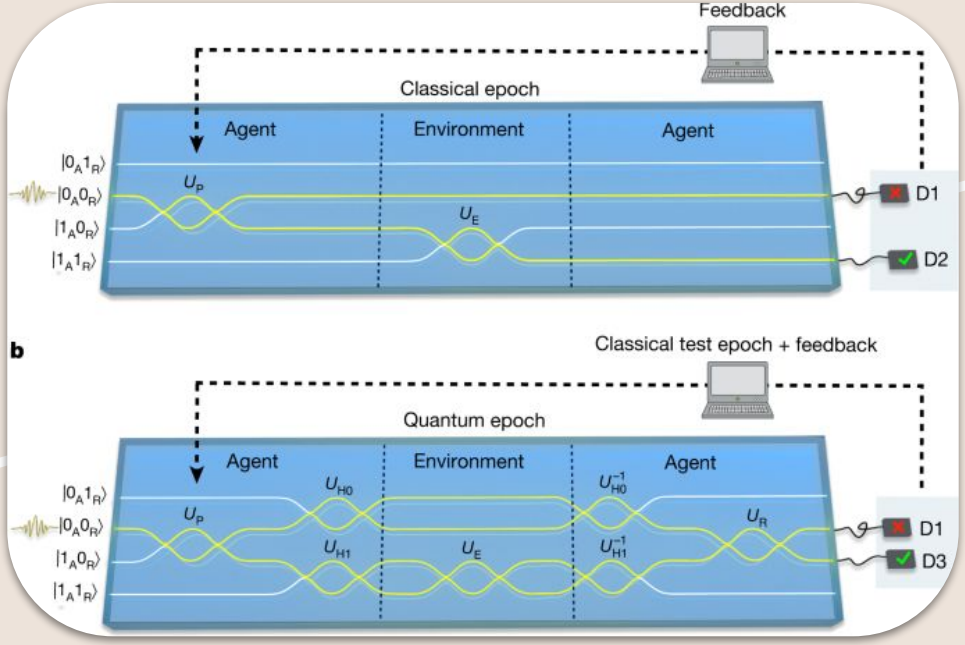


$$\frac{\pi}{2} = (2N_{optimal} + 1)\,\theta$$

$$N_{optimal} = \frac{\pi - 2\theta}{4\theta}$$

$$with\ \theta = arcsin(\beta)$$

Quantum Enhanced Reinforcement Learning



Setting

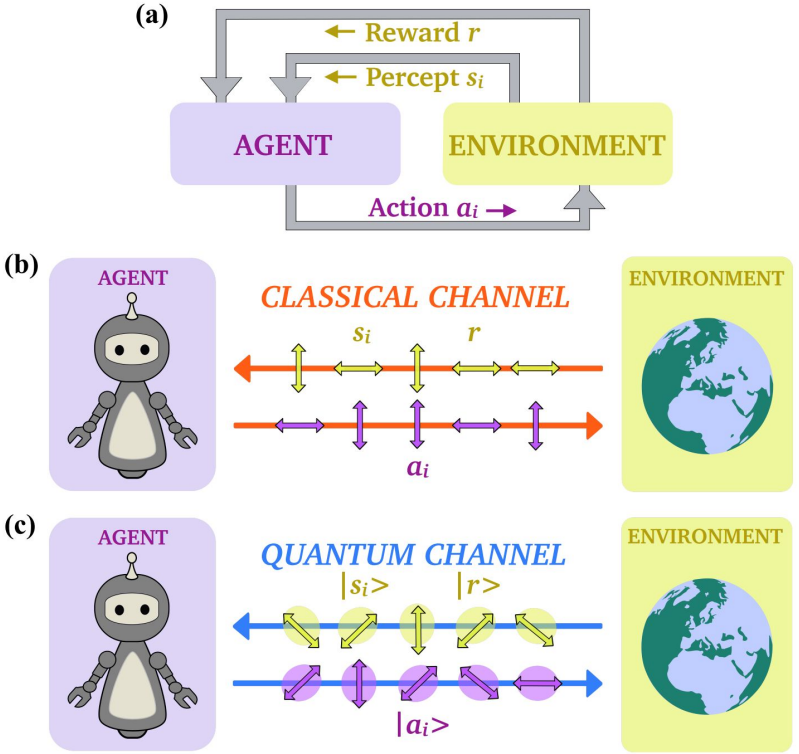
An agent can interact in two ways with the environment: classicaly and quantumly.

In the classical epoch, the agent prepare the state $|a\rangle|0\rangle$, in which the action a is sampled from its policy π with a probability $p(a)$. The agent then receives the reward with a probability equal to $\varepsilon = |\beta|^2$.

During a quantum epoch the agent prepares the state $|\psi\rangle_A|-\rangle_R$, where

$$|\psi\rangle_A = \sum_a \sqrt{p(a)}|a\rangle = cos(\xi)|lose\rangle_a + sin(\xi)|win\rangle_a = \alpha|0\rangle + \beta|1\rangle$$

represents the superposition of all actions with their probabilities and $|-\rangle_R = \frac{\sqrt{2}}{2}(|0\rangle_R + |1\rangle)_R$.



Setting

The environment applies the $Oracle$ Unitary to the prepared state

$$U_e|a\rangle_A|0\rangle_R = \begin{cases} |a\rangle_A|1\rangle_R & \text{if } r(a) > 0 \\ |a\rangle_A|0\rangle_R & \text{if } r(a) = 0 \end{cases} \tag{1}$$

which results in a bit flip of the winning state vector's phase

$$U_e|\psi\rangle_A|-\rangle_R = [cos(\xi)|lose\rangle_a - sin(\xi)|win\rangle_a]|-\rangle_R$$

The Grover's algorithm diffusion part is then performed as a reflection of the initial state $|\psi\rangle_A$ as

$$U_R = 2|\psi\rangle\langle\psi|_A - I_A$$

leading to an increased probability of sampling the winning state in the next classical epoch equal to $sin^2(3\xi)$, which lead to an increase in the rate of rewards obtained by the agent in the following step.

Setting

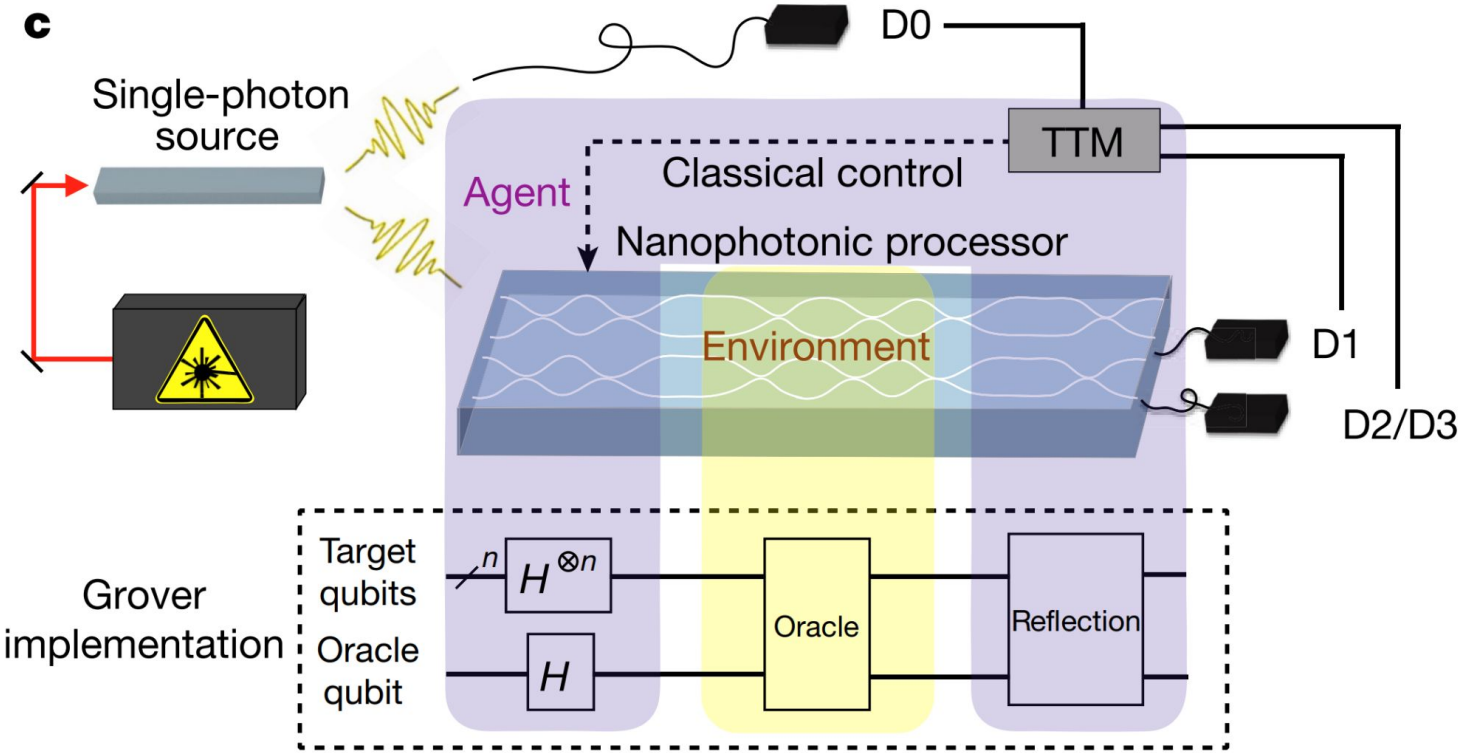
Due to the oscillating behaviour of Grover algorithm, the key of the approach relies on stopping the quantum strategy (alternating quantum and classical epoch) when the maximum probability is first reached and continue with the classical ones.

The optimal point in which this strategy change should happen is given by the following equation

$$sin^2(\xi) = \frac{sin^2(3\xi)}{2}$$

whose solution represents the value of ξ when the winning probability in the classical strategy is equal to half of the winning probability in the quantum strategy.

Setting


c
D0
Single-photon source
TTM
Agent
Classical control
Nanophotonic processor
Environment
D1
D2/D3
Grover implementation
Target qubits
Oracle qubit
n
$H^{\otimes n}$
H
Oracle
Reflection

Implementation - Agent

```
class Agent:

    def __init__(self, epsilon):
        self.reward_count = 0
        self.epsilon = epsilon

    def get_reward(self):
        self.reward_count += 1
        self.epsilon = (1+2*self.reward_count)/(100+2*self.reward_count)
```

Implementation - World

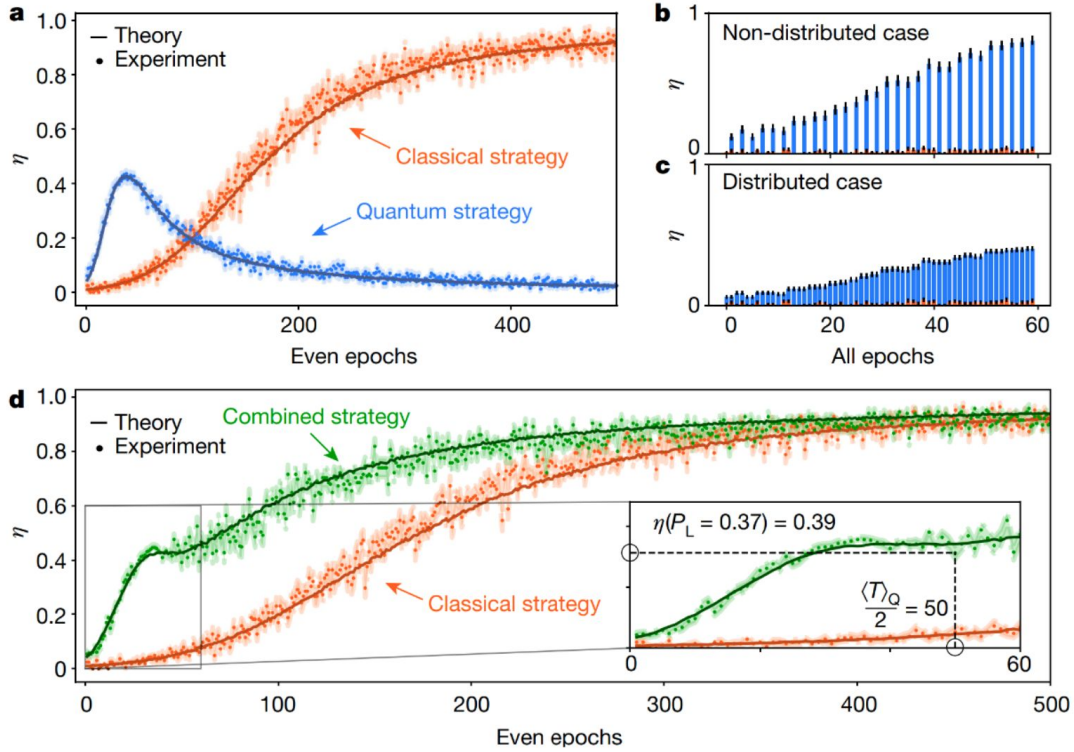
```python
class World:

    def __init__(self, initial_epsilon, combined_strategy_thresh=0.396):
        self.initial_epsilon = initial_epsilon
        self.circuit = self.init_circuit(initial_epsilon)
        self.combined_strategy_thresh = combined_strategy_thresh
        self.stop_quantum = False

    def combined_strategy(self, agent):
        epsilon = agent.epsilon
        reward_one = None

        if not self.stop_quantum:
            new_epsilon = self.quantum_epoch(epsilon)
        else:
            epsilon = agent.epsilon
            reward_one = self.classical_epoch(epsilon)
            if reward_one == 1:
                agent.get_reward()
            new_epsilon = epsilon

        reward_two = self.classical_epoch(new_epsilon)
        if reward_two == 1:
            agent.get_reward()

        if agent.epsilon > self.combined_strategy_thresh:
            self.stop_quantum = True

        if reward_one:
            return [reward_one, reward_two]
        else:
            return [reward_two/2, reward_two/2]
```

```python
def classical_strategy(self, agent):
    reward_one= self.classical_epoch(agent.epsilon)
    if reward_one == 1:
        agent.get_reward()

    reward_two = self.classical_epoch(agent.epsilon)
    if reward_two == 1:
        agent.get_reward()

    return [reward_one, reward_two]

def quantum_strategy(self, agent):
    new_epsilon = self.quantum_epoch(agent.epsilon)
    reward = self.classical_epoch(new_epsilon)
    if reward == 1:
        agent.get_reward()
    return [reward/2, reward/2]

def classical_epoch(self, epsilon):
    return np.random.choice(np.arange(0, 2), p = [1-epsilon, epsilon])

def quantum_epoch(self, epsilon):
    self.circuit = self.init_circuit(epsilon)
    new_epsilon = Statevector.from_instruction(self.circuit).data.real[1]**2
    return new_epsilon
```

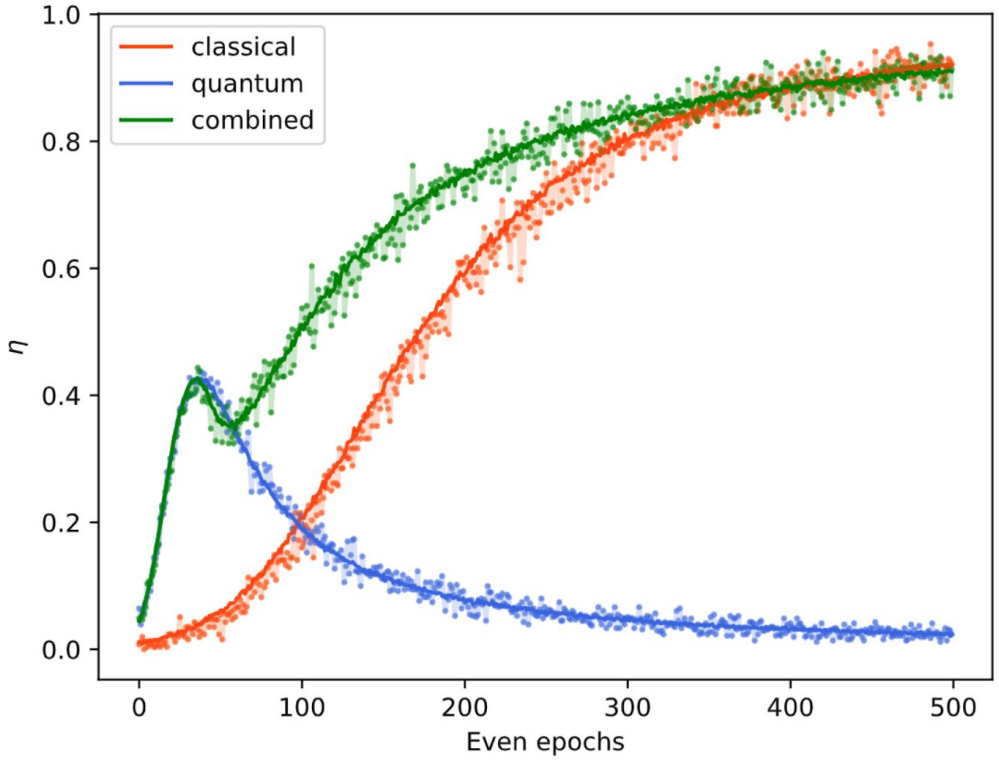
Implementation - World

```
def init_circuit(self, epsilon):
    circuit = QuantumCircuit(1)
    circuit.initialize([math.sqrt(1-epsilon), math.sqrt(epsilon)], 0)
    circuit.z(0)
    circuit.append(self.diffusion(epsilon), [0])
    return circuit

def diffusion(self, epsilon):
    a = Statevector([math.sqrt(1-epsilon),
                     math.sqrt(epsilon)])
    density = DensityMatrix(2*np.outer(a, a)-np.identity(2))
    c = UnitaryGate(density)
    return c
```

Results – Saggio et al.



Results – Caldarella



Thank you!

Questions?

Contact

simone.caldarella@studenti.unitn.it

simone.caldarella98@gmail.com