

Introduction to QSim



qctoolkit.in

qcworkbench@cdac.in

vivekn@cdac.in

PPT Link

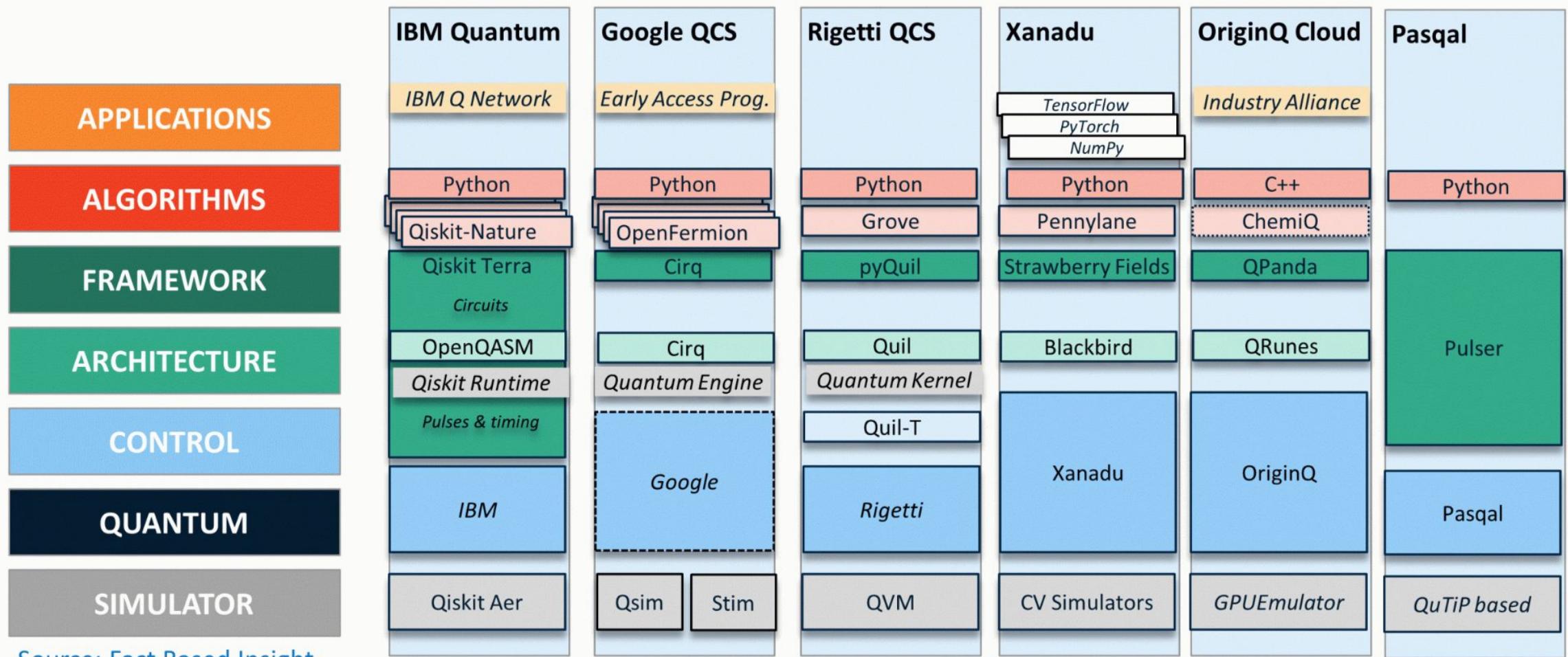
<https://tinyurl.com/qsim2024>

What it Offers?

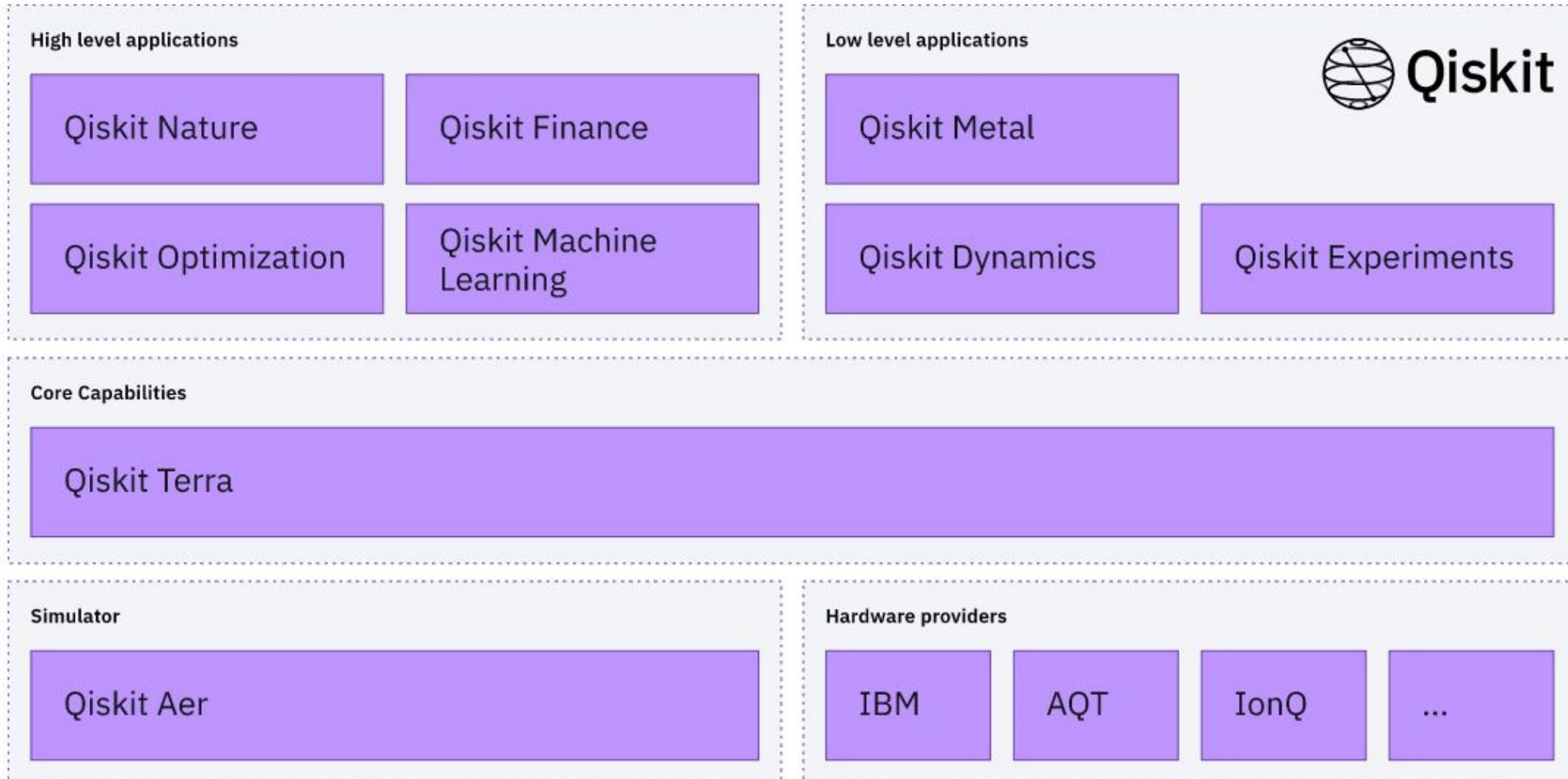
- **QSim** is a robust QC Simulator integrated with a GUI based Workbench
- Students / Researchers can create Quantum Circuits and Quantum Programs and view the simulated outputs

- QSim simulator is an open-source software written in Python, which is added as a new backend to IBM's Qiskit platform
- It extends the existing Qiskit capability, while retaining the convenience (e.g. portability, documentation, graphical interface) of the Qiskit format
- Simulator : arxiv.org/abs/1908.05154
- <https://github.com/indian-institute-of-science-qc/qiskit-aakash>
- [dm_simulator_user_guide](#)

Early gate-model full-stack players



Qiskit is open-source software for working with quantum computers at the level of circuits, pulses, and algorithms



Qiskit Software	Version
qiskit-terra	0.22.2
qiskit-aer	0.11.1
qiskit-ibmq-provider	0.19.2
qiskit	0.39.2
qiskit-nature	0.4.5
qiskit-finance	0.3.4
qiskit-optimization	0.4.0
qiskit-machine-learning	0.4.0
System information	
Python version	3.8.14
Python compiler	GCC 9.4.0
Python build	default, Sep 7 2022 14:28:32
OS	Linux
CPUs	2
Memory (Gb)	6.78125
Fri Nov 04 02:27:15 2022 UTC	

When using Qiskit a user workflow nominally consists of following four high-level steps:

- **Build**: Design a quantum circuit(s) that represents the problem you are considering.
- **Compile**: Compile circuits for a specific quantum service, e.g. a quantum system or classical simulator.
- **Run**: Run the compiled circuits on the specified quantum service(s). These services can be cloud-based or local.
- **Analyze**: Compute summary statistics and visualize the results of the experiments.

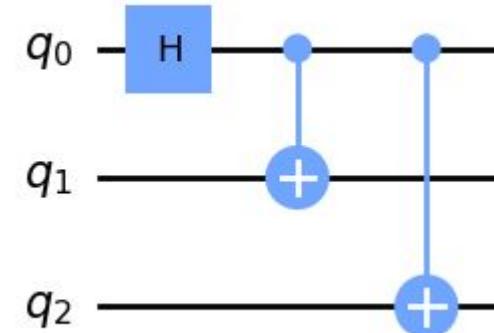
The basic element needed for your first program is the **QuantumCircuit**

This is taken care by the **Qiskit Terra**

```
# Create a Quantum Circuit acting on a quantum register of three qubits
circ = QuantumCircuit(3)
```

```
# Add a H gate on qubit 0, putting this qubit in superposition.
circ.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1, putting
# the qubits in a Bell state.
circ.cx(0, 1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 2, putting
# the qubits in a GHZ state.
circ.cx(0, 2)
```

```
import numpy as np
from qiskit import *
```



Simulating circuits using **Qiskit Aer**

```
# Import Aer
from qiskit import Aer

# Run the quantum circuit on a statevector simulator backend
backend = Aer.get_backend('statevector_simulator')
```

```
# Create a Quantum Program for execution
job = backend.run(circ)
```

```
outputstate = result.get_statevector(circ, decimals=3)
print(outputstate)

Statevector([0.707+0.j, 0.    +0.j, 0.    +0.j, 0.    +0.j, 0.    +0.j,
            0.    +0.j, 0.    +0.j, 0.707+0.j],
dims=(2, 2, 2))
```

1 - Program on QSim /Density Matrix simulator

```
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute, BasicAer
import numpy as np

#Options & Noise goes here - Don't change options variable name & block
options = {

    "rotation_error": {'rx':[1.0, 0.0], 'ry':[1.0, 0.0], 'rz':[1.0,0.0]},
    "tsp_model_error": [1.0, 0.0],
    "thermal_factor": 1.0,
    "decoherence_factor": 1.0,
    "depolarization_factor": 1.0,
    "bell_depolarization_factor": 1.0,
    "decay_factor": 1.0,
}
```

3 – Running Quantum Circuit on QSim and Getting Results

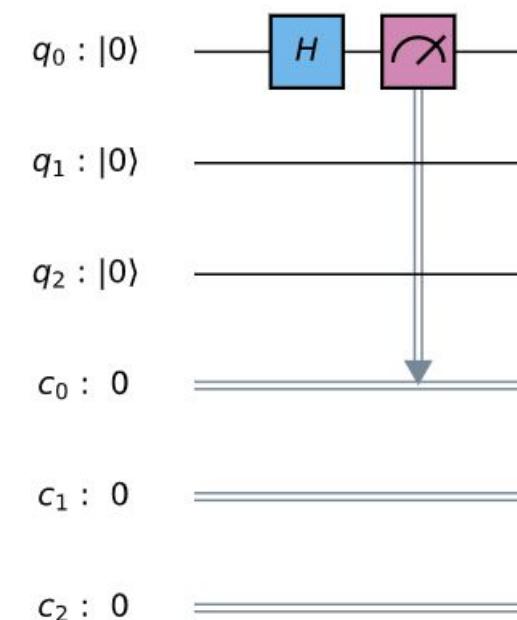
```
backend = BasicAer.get_backend('dm_simulator')
job = execute(qc, backend=backend, **options)
job_result = job.result()
print(job_result.results[0].data.densitymatrix)
```

2 - Circuit Creation same as provided by Qiskit Terra

```
qc = QuantumCircuit()
q = QuantumRegister(3, 'q')
c = ClassicalRegister(3, 'c')

qc.add_register(q)
qc.add_register(c)

qc.h(q[0])
qc.measure(q[0], c[0])
```



Implementation

- Standard formulation of quantum mechanics, states are vectors in a Hilbert space and evolve by unitary transformations, $|\psi\rangle \rightarrow U|\psi\rangle$. This evolution is deterministic, continuous and reversible.
- It is appropriate for describing the pure states of a closed quantum system, but is insufficient for describing the mixed states that result from interactions of an open quantum system with its environment
- The most general description of a quantum system is in terms of its density matrix ρ

- Quantum systems are highly sensitive to disturbances from the environment; even necessary controls and observations perturb them.
- The available, and upcoming, quantum devices are noisy, and techniques to bring down the environmental error rate are being intensively pursued
- It is necessary to come up with error-resilient system designs, as well as techniques that validate and verify the results

- This era of noisy intermediate scale quantum systems has been labeled **NISQ**
- Such systems are often special purpose platforms, with limited capabilities
- They roughly span devices with 10- 100 qubits, 10-1000 logic operations, limited interactions between qubits, and with no error correction since the fault-tolerance threshold is orders of magnitudes away

- A quantum computation may suffer from many sources of error
 - Imprecise initial state preparation
 - Imperfect logic gate implementation
 - Disturbances to the data in memory
 - Error-prone measurements

So a realistic quantum simulator would have to include all of them with appropriate probability distributions

QSim: Quantum Computer Simulator Toolkit



Simulate quantum circuits: Simulation of Quantum circuits with custom parameters.



Quantum Noise: Realistic simulator considering effects of noise



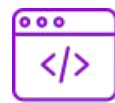
Intuitive UI: Intuitive UI/UX helps users to conceptualize and create quantum programs



Examples & Help: Online help, solved examples and learning material.



Secured user management: Secure user management with options to save quantum programs/circuits



Code editor: Advanced Python code editor for Quantum Circuits.



QC Workbench Features

- Interactive Python code editor with IDE features such as
 - Syntax highlighting
 - Parsing and error checking
 - Autocompleting keywords and variables
 - Code folding and unfolding
 - Automatic braces detecting and closing
 - Search and replace keywords.

- Build and visualize Quantum Circuits

</> Python Editor

```
 6 "rotation_error": [1.0, 1.0], "y": [1.0, 1.0], "z": [1.0, 1.0]
 7 "tsp_model_error": [1.0, 1.0],
 8 "thermal_factor": 1.0,
 9 "decoherence_factor": 1.0,
10 "depolarization_factor": 1.0,
11 "bell_depolarization_factor": 1.0,
12 "decay_factor": 1.0,
13 }
14 ######Write your code after this line#####
15 qreg_q = QuantumRegister(2, 'q')
16 creg_c = ClassicalRegister(2, 'c')
17 circuit = QuantumCircuit(qreg_q, creg_c)
18
19 backend = BasicAer.get_backend('dm_simulator')
20 job = execute(circuit, backend=backend, **options)
21 job_result = job.result()
```

</> Quantum Circuit

$q_0 : |0\rangle$ —————

$q_1 : |0\rangle$ —————

$c_0 : 0$ —————

$c_1 : 0$ —————

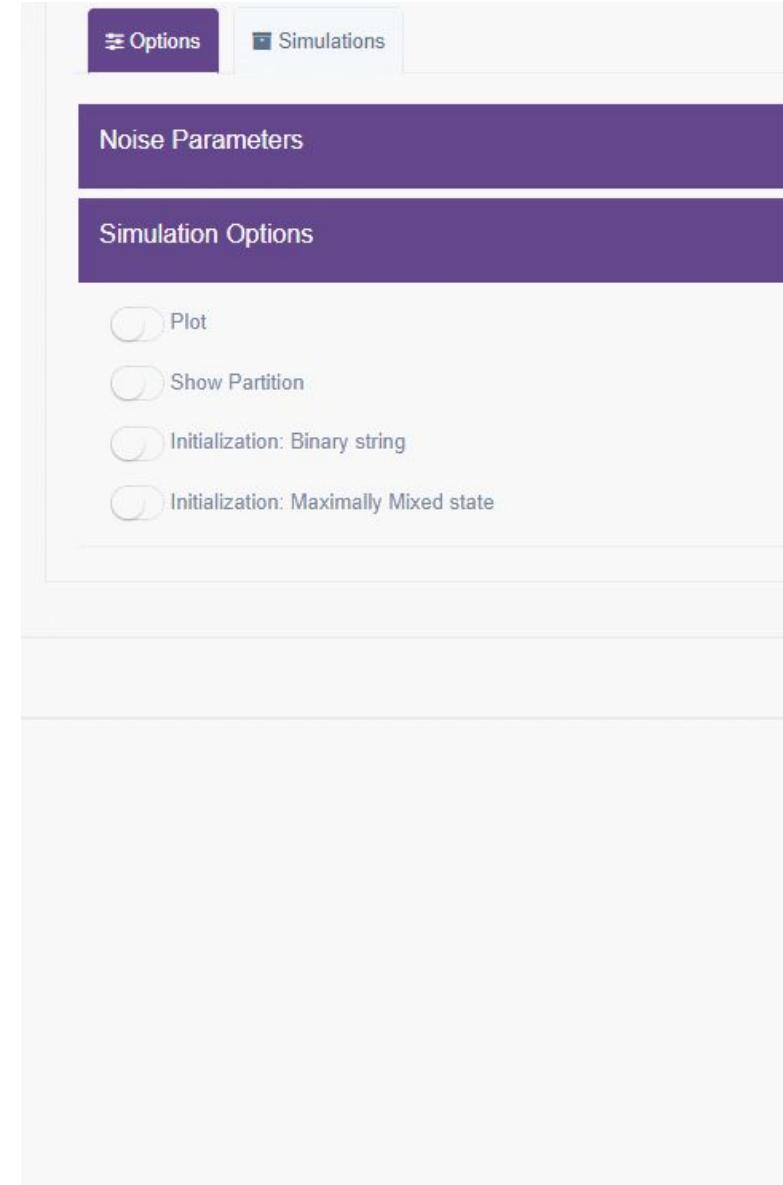
QC Workbench Features

- **Induce Noise parameters**

- Rotation Error
- TSP Error
- Decay factor
- Decoherence
- Depolarization Factor
- Thermal Factor
- Bell Depolarization Factor

- **Customize simulation options**

- Enable/Disable Plot
- Enable/Disable circuit partitioning
- Circuit initialization options.



QC Workbench Features

- Support for multiple measurement options
 - Single QuBit measurement
 - Expectation measurement
 - Ensemble measurement



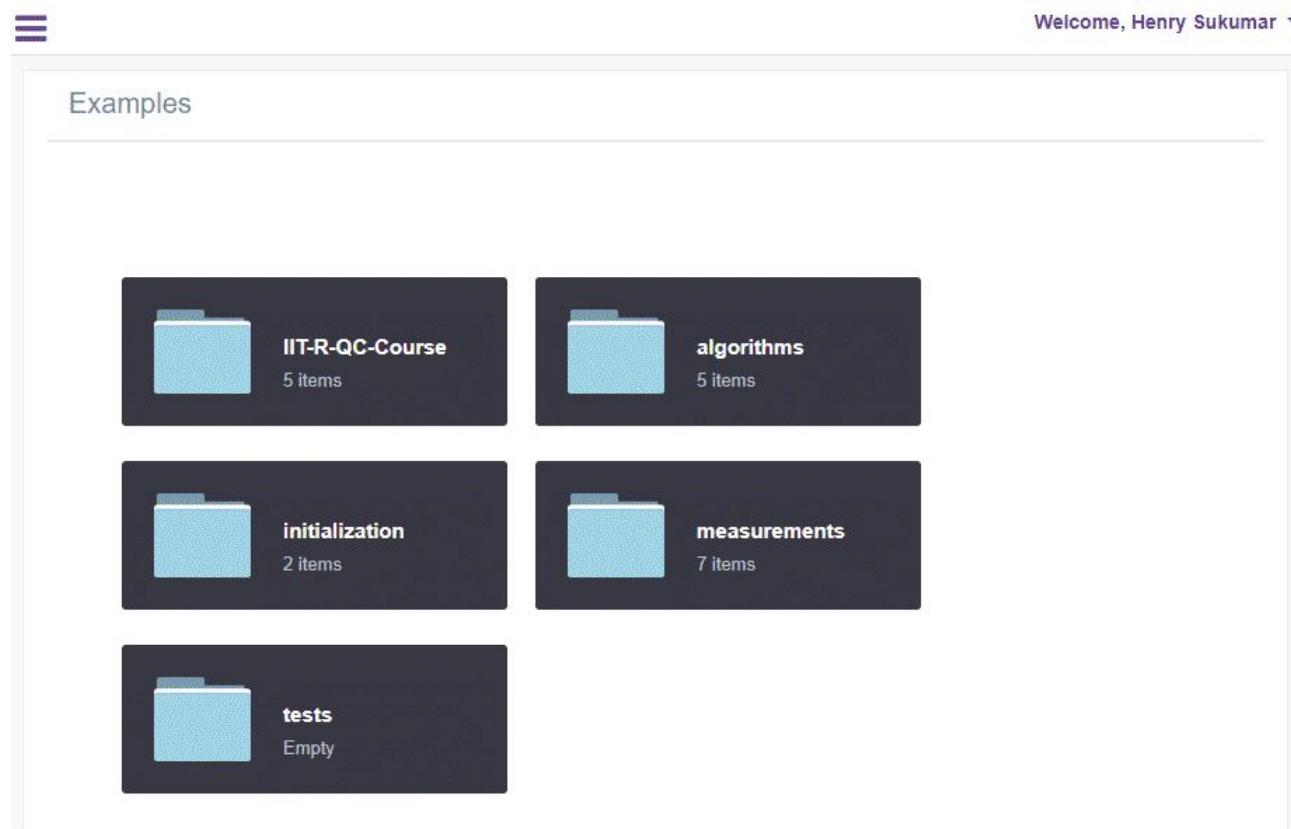
The screenshot shows a Python code editor interface for a quantum circuit. The code is as follows:

```
16 ##### Write your code after this line#####
17 qc = QuantumCircuit()
18 q = QuantumRegister(1, 'q')
19 c1 = ClassicalRegister(2, 'c1')
20
21 qc.add_register(q)
22 qc.add_register(c1)
23
24 qc.h(q[0])
25 qc.measure(q[0], c1[0])
26
27 backend = BasicAer.get_backend('dm_simulator')
28 job = execute(qc, backend=backend, **options)
29 job_result = job.result()
30 print(job_result['results'][0]['data']['densitymatrix'])
31
```

The code defines a QuantumCircuit with one qubit register and two classical bit registers. It performs a Hadamard gate on the first qubit and measures it into the first classical bit. The result is printed as a density matrix.

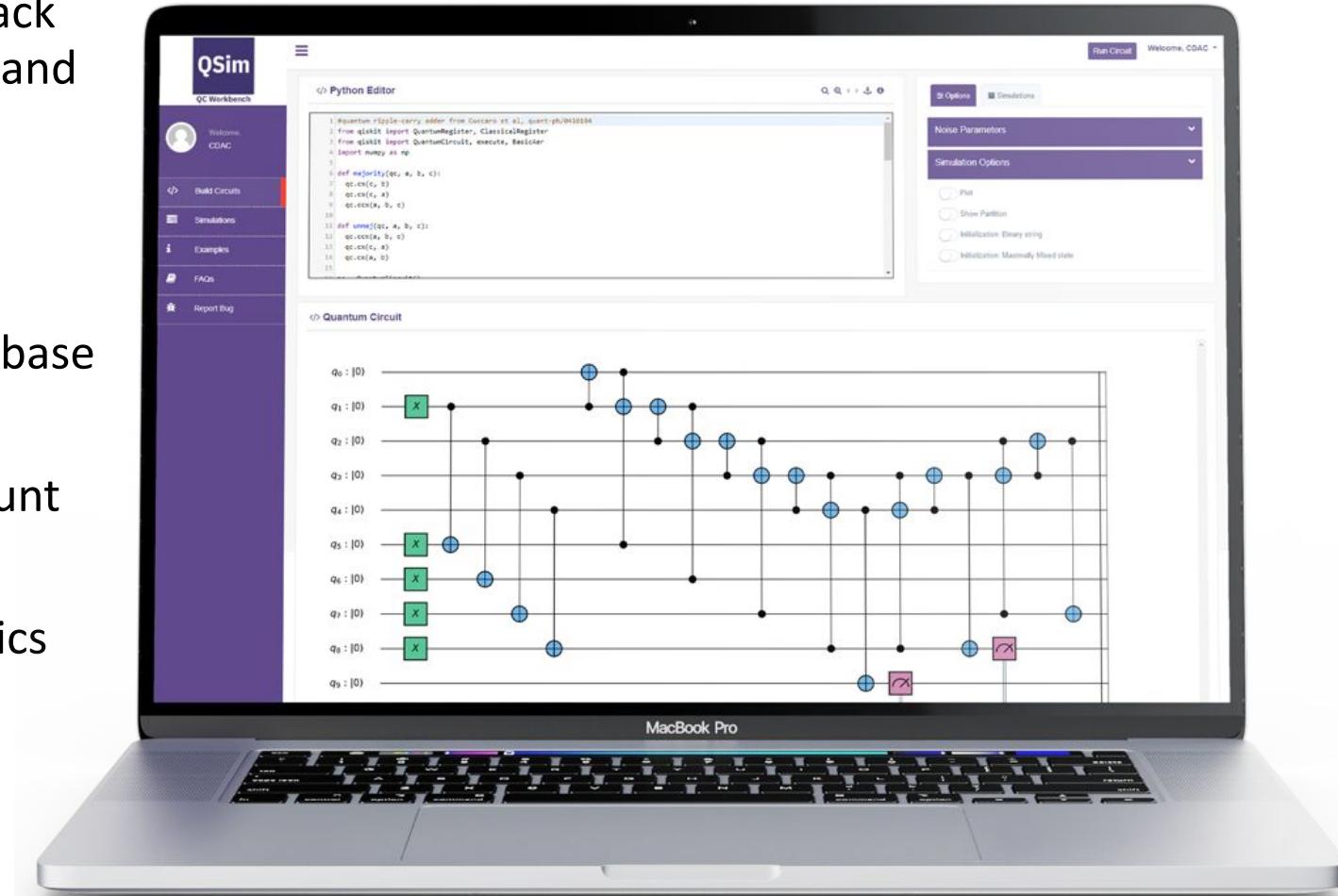
QC Workbench Features

- Pre-loaded Quantum examples and algorithms
 - Deutsch-Jozsa Algorithm
 - Hubbard model
 - QFT
 - Grover's algorithm
 - Ripple Carry Adder



QC Workbench Features

- Submit simulations, track progress, fetch results and accounting
- Plot and visualization histograms
- Integrated Knowledge base & Guides
- Secure Login and account management
- User simulation statistics
- Bug reporting.



Login Screen (qctoolkit.in)



[Home](#) [About](#) [Knowledge Base](#) [FAQ](#) [Log In](#)

Members

Login

Username or Email Address

Password

Remember Me

Log In

- [Register](#)
- [Lost your password?](#)



Privacy - Terms

Interactive UI

QC Workbench

Welcome, vivekn

Build Circuits

Simulations

Examples

FAQs

Report Bug

Run Circuit

Welcome, vivekn

</> Python Editor

```
121 ##Option & Noise parameters goes here - Don't change options variable name
122 options = {
123
124     'plot': True,
125     "rotation_error": {'rx':[1.0, 1.0], 'ry':[1.0, 1.0], 'rz':[1.0,1.0]},
126     "tsp_model_error": [1.0, 1.0],
127     "thermal_factor": 1.0,
128     "decoherence_factor": 1.0,
129     "depolarization_factor": 1.0,
130     "bell_depolarization_factor": 1.0,
131     "decay_factor": 1.0,
132 }
133 run = execute(qc,backend,**options)
134 result = run.result()
135 bell_basis = result['results'][0]['data']['bell_probabilities01']
136 print (bell_basis).
```

</> Quantum Circuit

q₀₀ : |0>

q₀₁ : |0>

q₀₂ : |0>

c₀₀ : 0

c₀₁ : 0

c₀₂ : 0

Options

Simulations

Noise Parameters

Simulation Options

Plot

Show Partition

Initialization: Binary string

Initialization: Maximally Mixed state

Welcome,
Vivek

</> Build Circuits

≡ Simulations

i Examples

FAQs

About dm_sim

Open in Notebook

Report Bug/Feedback

Analytics



Welcome, Vivek ▾

JobID:139521

Refresh Export Edit to Re-Submit

Run Details

Status:	Submitted On:	Job Name:	Time Taken (secs):
COMPLETED	2024-02-26 10:40:13	untitled.py	00:00:34

Result:

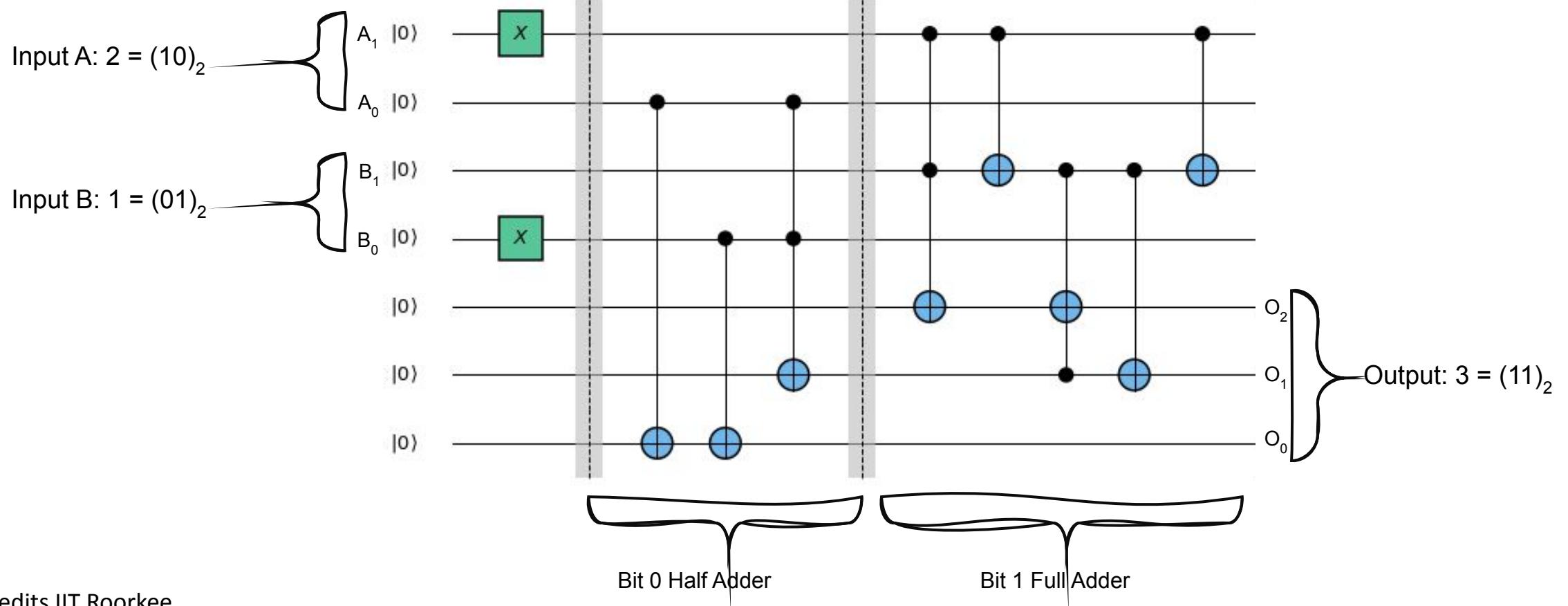
Output Error Code Circuit

```
=====
SLURM_CLUSTER_NAME = paramutkarsh
SLURM_ARRAY_JOB_ID =
SLURM_ARRAY_TASK_ID =
SLURM_ARRAY_TASK_COUNT =
SLURM_ARRAY_TASK_MAX =
SLURM_ARRAY_TASK_MIN =
SLURM_JOB_ID = 139521
SLURM_JOB_NAME = qsim
SLURM_JOB_NODELIST = cn107
SLURM_JOB_UID = 21035
SLURM_JOB_PARTITION = standard
SLURM_TASK_PID = 41677
SLURM_CPUS_ON_NODE = 1
SLURM_NTASKS = 1
SLURM_TASK_PID = 41677
=====

Job ran on IISC DM Simulator at Mon Feb 26 05:01:14 UTC 2024
[[0.5+0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0.5+0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j]]
```



Quantum Circuit for 2-bit Ripple Adder



Installing QSim Locally(On Linux)

- #Install rust compiler dependency for new version of qiskit

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

- #Install miniconda if previously not installed

```
mkdir -p ~/miniconda3
```

```
wget
```

```
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86\_64.sh
```

```
-O
```

```
~/miniconda3/miniconda.sh
```

```
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
```

```
rm -rf ~/miniconda3/miniconda.sh
```

- #can add the miniconda activation in bash by default

```
~/miniconda3/bin/conda init bash
```

- #Restart the shell so that the environment for miniconda take place or activate the env

```
source ./miniconda3/bin/activate
```

- #Create a conda environment for installing GPU version of QSim simulator backend

```
conda create -n qsimenv python=3.8.5
```

```
conda activate qsimenv
```

- # Clone the stable version of QSim

```
git clone  
-b terra_upgrade https://github.com/indian-institute-of-science-qc/qiskit-aakash.git
```

```
cd qiskit-aakash
```

```
git checkout -b dev bea98fbff86c234c0b1990add17493a1e86917cb
```

```
python -m pip install -e .
```

```
python -m pip install jupyterlab
```

- # Launch jupyter lab on terminal

```
jupyter lab
```

Additional Resource For QC

Here are the links for the 2020 Quantum Computing Course conducted by IIT Roorkee

<https://github.com/deadbeatfour/quantum-computing-course>

<https://www.youtube.com/playlist?list=PLyEHBEYaB52XCMH9mMHo5MAzGcZkcdVGB>

Here are the links for the 2023 Quantum Computing Course conducted by IIT Roorkee

https://github.com/gitkarma/quantum_computing_course2023

https://www.youtube.com/playlist?list=PLt_nrfusQeEdYto6Qh2hkD7EcCQ-7QHFk

Here are the links for the 2025 Quantum Computing course

https://github.com/gitkarma/quantum_computing_course2025

https://www.youtube.com/playlist?list=PLt_nrfusQeEc-5tBqiQkmt70Aeu_zNiNT

[Introduction to Quantum Computation by IISc Bangalore](#)

https://www.youtube.com/watch?v=-t8IvCAzeKY&list=PLNrdQx59gmEWV7eLbqDWel_B72-0MXiyt

- Representing the state of qubits using the state-vector notation
- Convenient when dealing with states that can always be expressed as a linear combination of basis states
- There are many practical scenarios in which the state of our qubits cannot be written down as linear combinations in a given basis
- But instead, must be expressed in terms of ensembles (statistical mixtures) of multiple states
- Density matrix is an alternative way of expressing quantum states

1. Pure States

Pure states are those for which we can precisely define their quantum state at every point in time. For example, if we initialize the single qubit $|q\rangle$ in state $|0\rangle$, and apply a Hadamard gate, we know our final state will be:

$$|q\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle$$

We understand that if we were to perform a measurement of this state, the outcome will be probabilistic. We will measure state $|0\rangle$ with 50% probability, state $|1\rangle$ with 50% probability. However, **before** performing any measurements we can say with 100% certainty that, if our qubit initialization process and our Hadamard gate are ideal, the resulting quantum state will always be $|+\rangle$. We therefore say that, since there is no uncertainty on what this quantum state will be, $|q\rangle$ is a pure state.

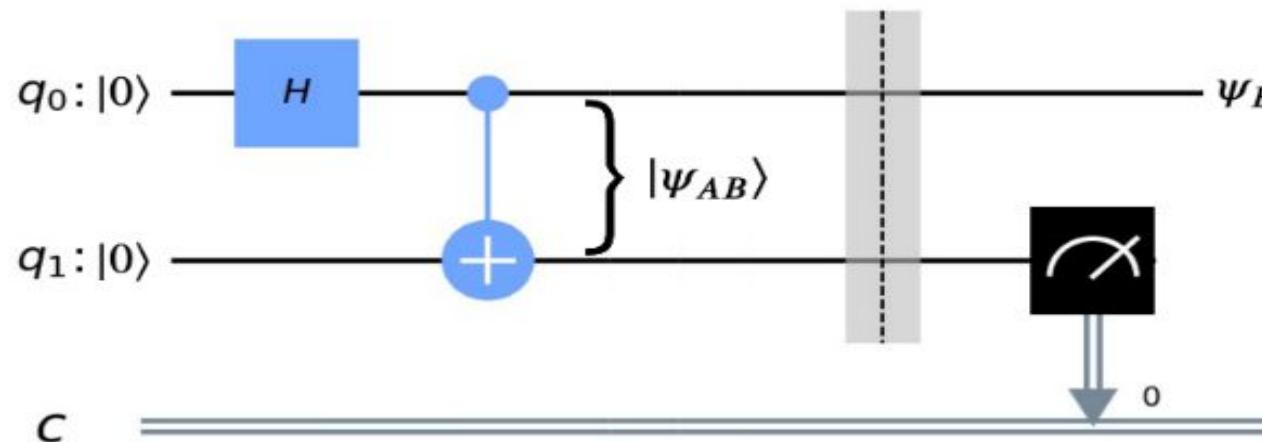
2. Mixed States

Mixed states are those that consist of statistical ensembles of different quantum states. This means that, unlike pure states, mixed states cannot be represented as linear superpositions of normalized state vectors. Let's first take a look at a simple example to explain what we mean by this.

Consider, once again, the two-qubit entangled state:

$$|\psi_{AB}\rangle = \frac{1}{\sqrt{2}}(|0_A0_B\rangle + |1_A1_B\rangle)$$

Here we have explicitly used the subscripts A and B to label the qubits associated with registers q_1 and q_0 , respectively. Now, let's assume that right after preparing our state $|\psi_{AB}\rangle$ we perform a measurement on register q_1 , as shown below:



We can now ask the question: What will the state on register q_0 be right after performing the measurement on register q_1 ?

As we learned in the [Multiple Qubits and Entangled States](#) section, we know that since qubits A and B are entangled, measuring a 0 in register q_1 implies that the quantum state in register q_0 will immediately project onto state $|0_B\rangle$:

$$\frac{1}{\sqrt{2}}(|0_A0_B\rangle + |1_A1_B\rangle) \xrightarrow{\text{measure}} |0_A\rangle|0_B\rangle \quad (\text{with probability of 50\%})$$

Similarly, measuring a 1 in register q_1 will project q_0 onto state $|1_B\rangle$:

$$\frac{1}{\sqrt{2}}(|0_A0_B\rangle + |1_A1_B\rangle) \xrightarrow{\text{measure}} |1_A\rangle|1_B\rangle \quad (\text{with probability of 50\%})$$

So how do we, in general, represent the final state in register q_0 (labeled as ψ_B in the diagram), not for a specific measurement outcome in q_1 , but for an arbitrary result of this measurement process?

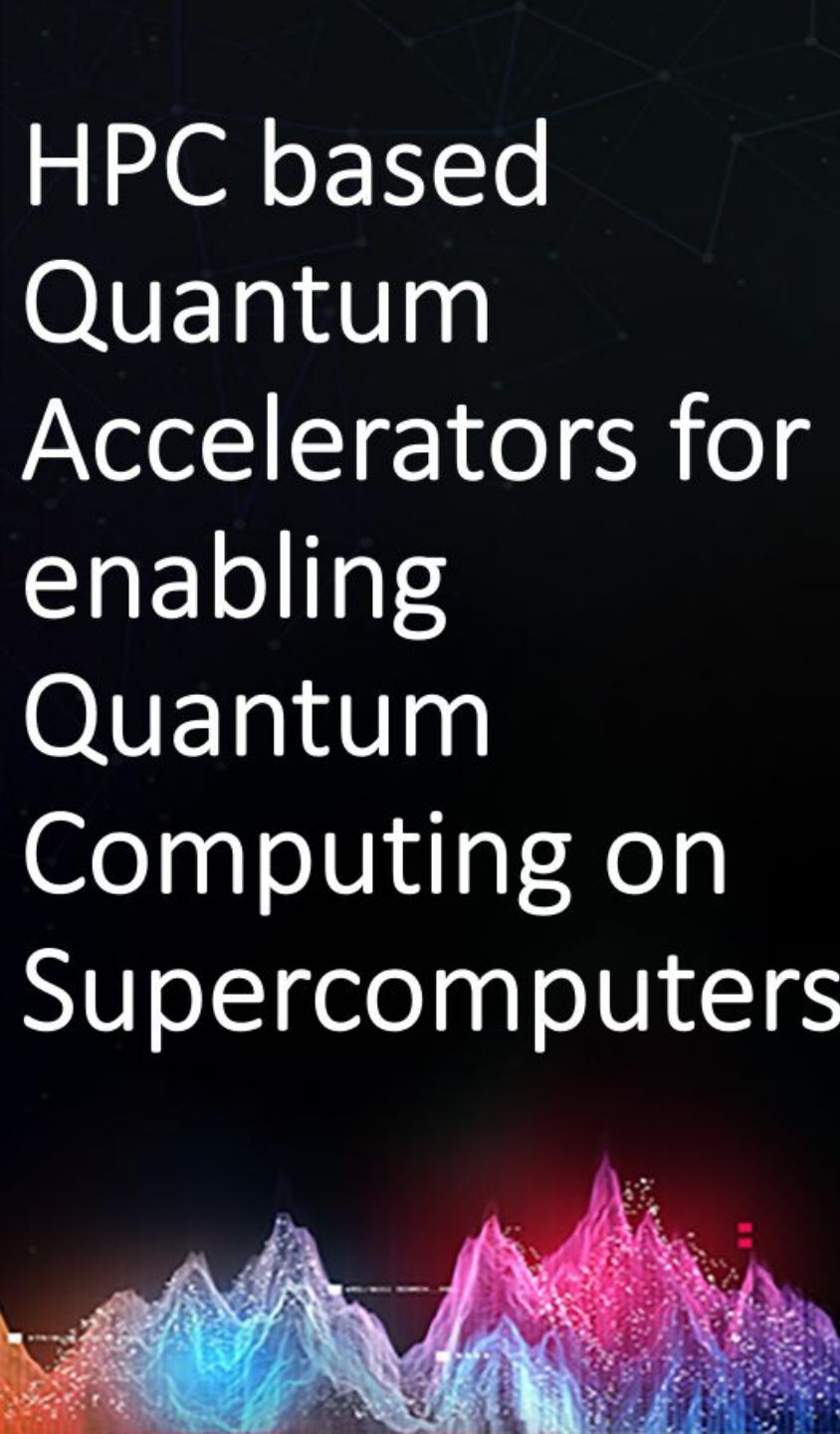
We know that after a measurement, ψ_B will be in state $|0_B\rangle$ with probability $1/2$, or in state $|1_B\rangle$ with probability $1/2$; however, ψ_B is **not** in a linear superposition of $|0_B\rangle$ and $|1_B\rangle$. In other words, ψ_B **cannot** be expressed as a state vector of the form $1/\sqrt{2}(|0_B\rangle + |1_B\rangle)$. Instead, we have to use a different notation to write down that ψ_B is rather an ensemble (not a quantum superposition) of the states $|0_B\rangle$ and $|1_B\rangle$, and whose outcome depends on what we measure on register q_1 .

We then call ψ_B a mixed state, which can be represented as an ensemble of states:

$$\{|\psi_{B_0}\rangle, |\psi_{B_1}\rangle\} = \{|0_B\rangle, |1_B\rangle\},$$

Country	Technology	Companies/ University
The United States of America	Superconducting based	IBM, Google, Rigetti
	Ion Trap-based	IonQ
	Photonics based	Psi- Quantum
	Neutral atoms based	ColdQuanta, Atom Computing QuEra Computing
	Semiconductor based	Princeton University
Canada	Photonics based	Xanadu
The United Kingdom	Ion Trap- based	Quantinuum, Universal Quantum
	Photonics based	Orca Computing, Tundrasystems Global
	Semiconductor based	Quantum Motion
Finland	Superconducting based	IQM
	Semiconducting based	QuTech
Germany	Ion Trap- based	ele Qtron
	Neutral atom based	Planqc
Austria	Ion trap- based	Alpine Quantum Technologies
France	Neutral atoms based	PasQal
	Photonics based	Quandela

HPC based Quantum Accelerators for enabling Quantum Computing on Supercomputers



Overall Objective: The objective is to build a Quantum Accelerators wherein an Accelerator card is integrated with HPC nodes to achieve computational tasks that are otherwise beyond the reach of the available Quantum Simulators. It is proposed to develop three testbed GPU/Vector/FPGA based and enable QSim/other open source simulator environment. Two deployment on Param Siddhi and Param Utkarsh and Cloud based Quantum Experience Centre.

Collaborating Centres: C-DAC Pune, Bengaluru, Hyderabad, Noida, Patna

CDAC Hyderabad

- Porting of QSim and other Open Source Quantum Simulators on difference Accelerator Platforms (GPU/VECTOR)
- Carry out Benchmarking on these platforms
- Dissemination and Knowledge Sharing



GPU CARD



VECTOR ENGINE CARD

QSim for VECTOR Processors

- QSim (dm_simulator) (simulate noisy quantum logic circuits using the density matrix formalism)
 - Current status (dm_simulator)- Available for GPU
 - Porting over vector processors and making it available over the HPC QA Testbed
 - Benchmarking the ported version
 - Upgrades and Support



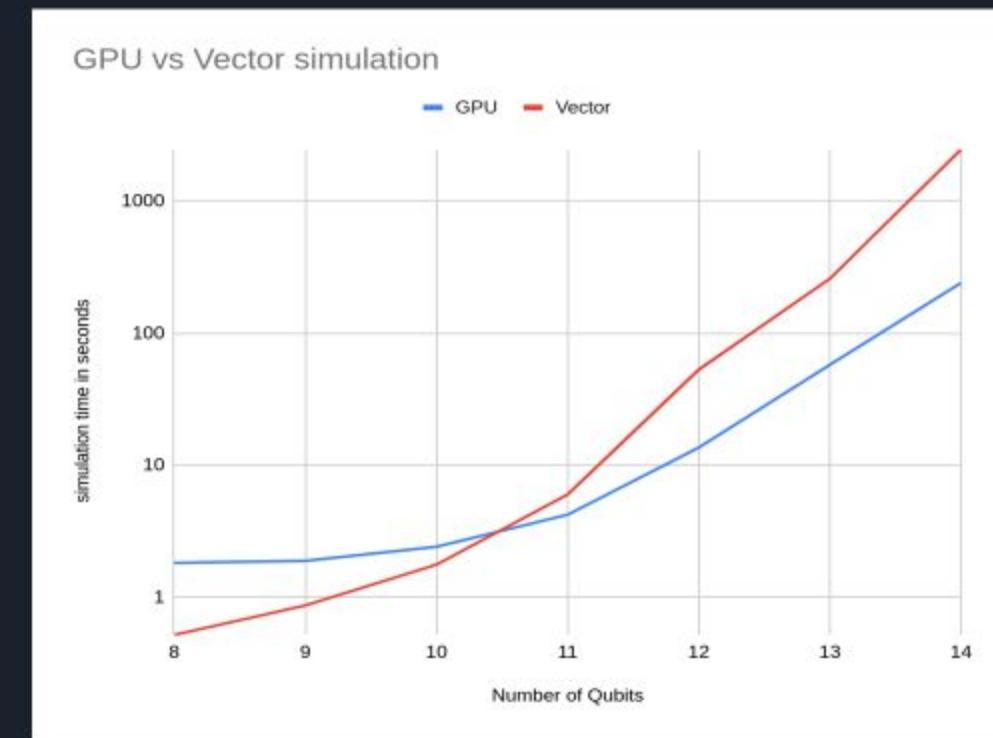
GPU CARD



VECTOR ENGINE CARD

QSim Simulation Timings (Quantum Fourier Transform)

Qubits	GPU	Vector
8	1.83	0.52
9	1.89	0.87
10	2.42	1.77
11	4.23	6.02
12	13.66	53.17
13	57.80	259.71
14	241.36	2474.26

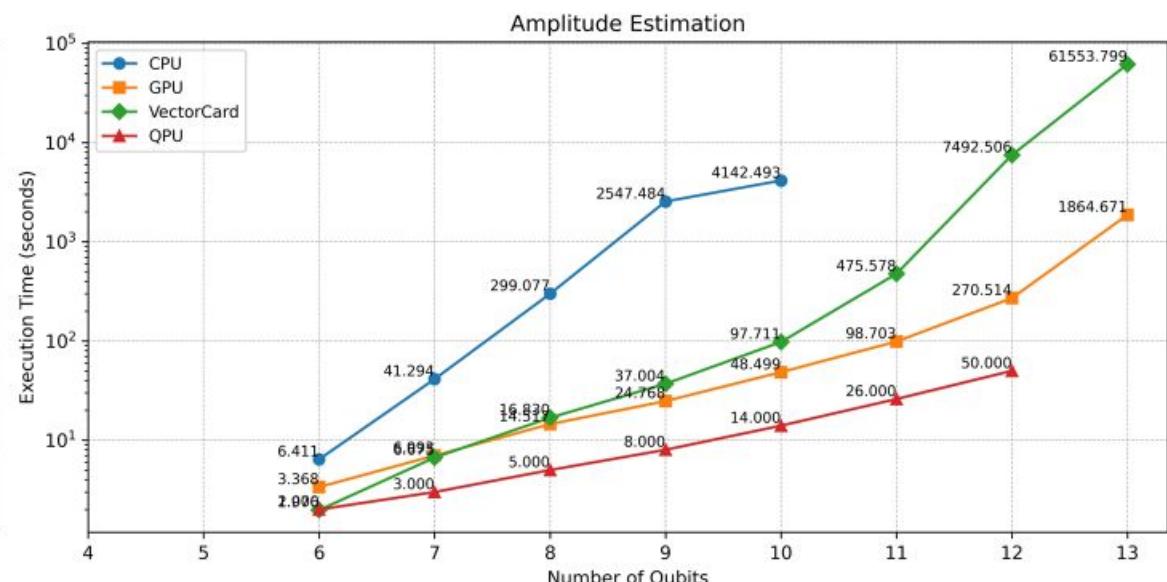
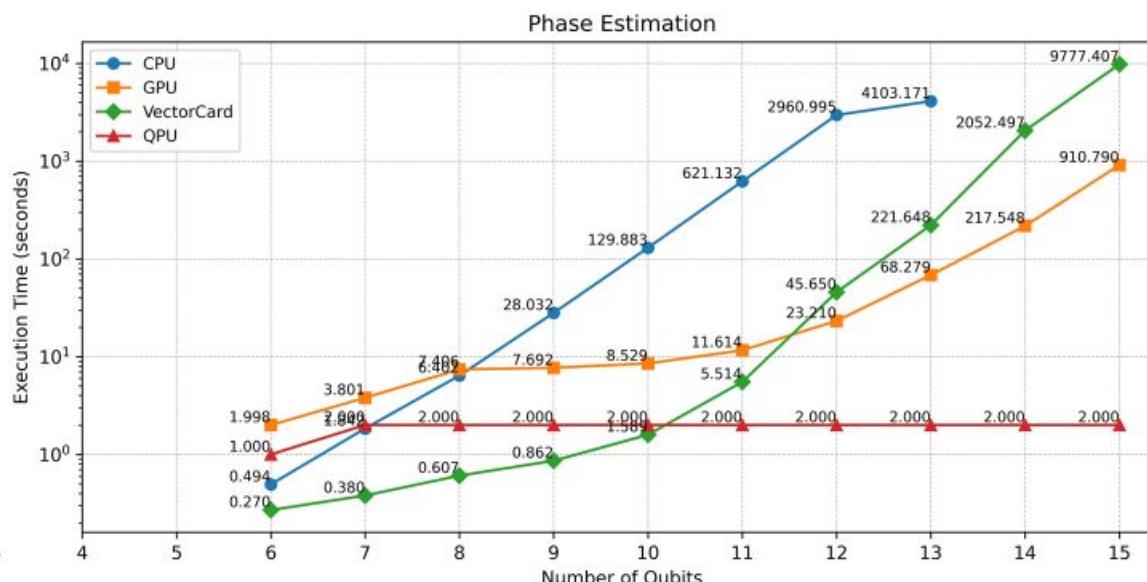
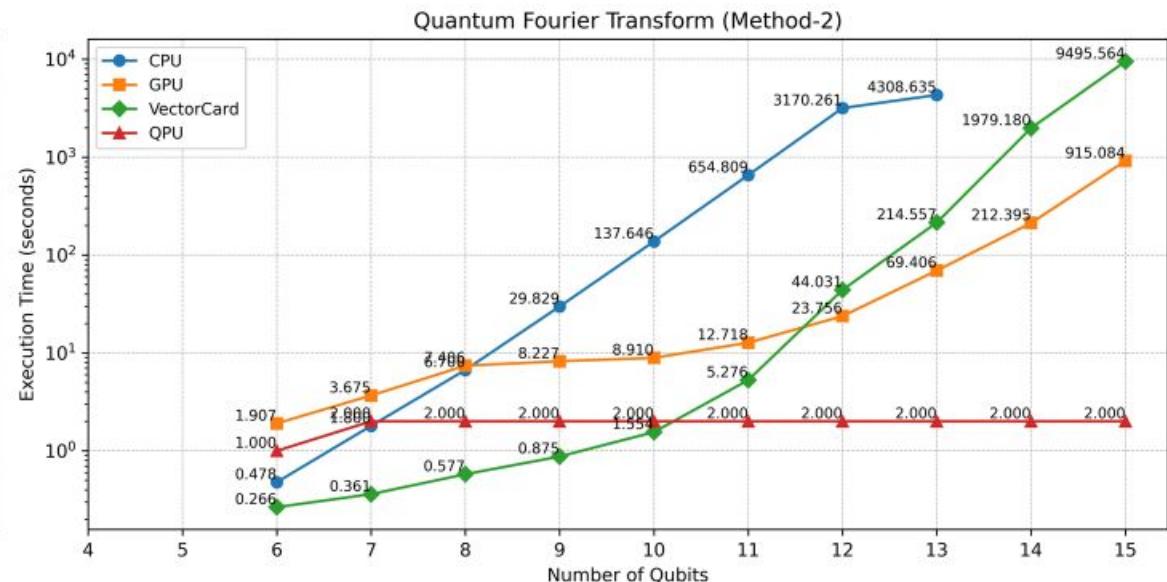
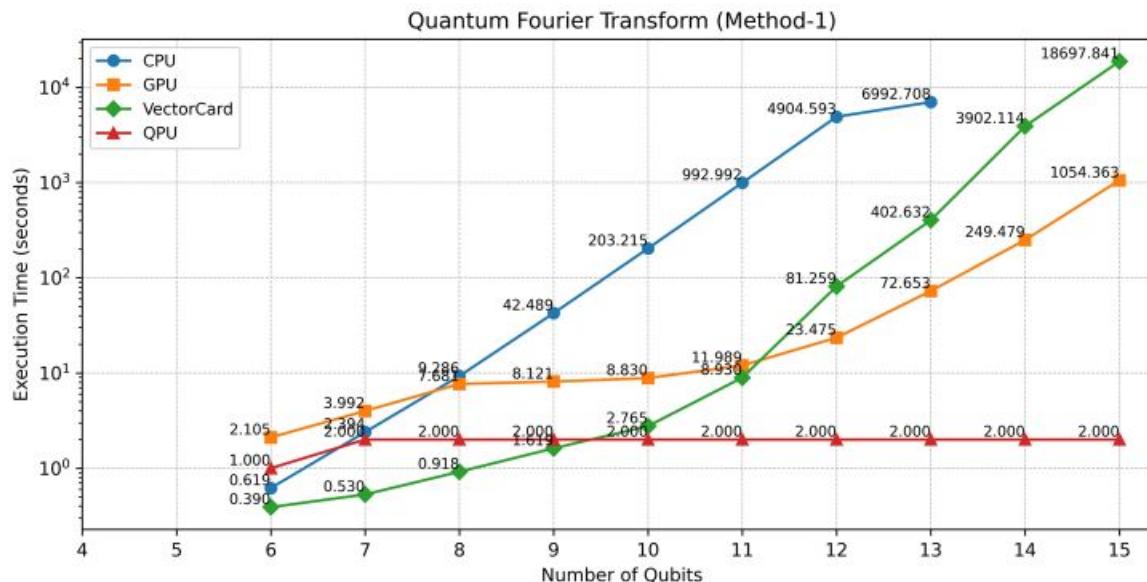


- **GPU:** Nvidia RTX A5000 (24GB)
- **CPU Cores:** 96
- **System RAM:** 256 GB

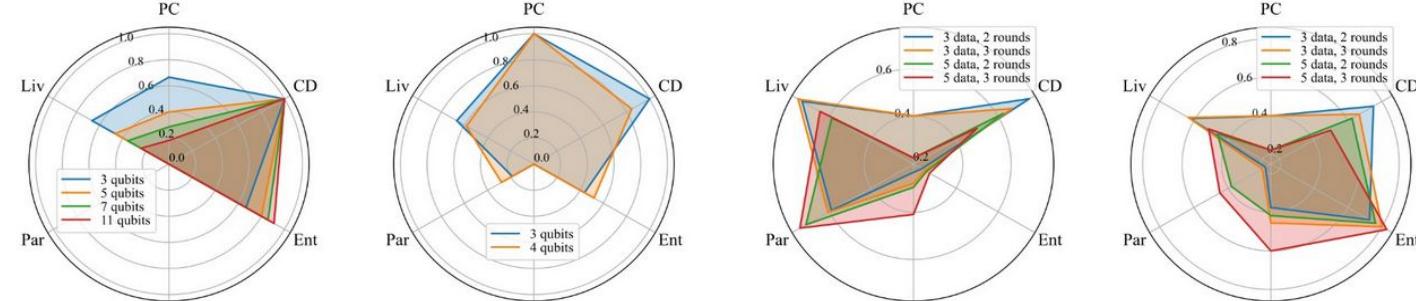
- **NEC Vector Engine:** VE20B Tsubasa SX-Aurora (48 GB)
- **CPU Cores:** 64
- **System RAM:** 512 GB

Execution Time Comparison across CPU, GPU, Vector and QPU (QC App Oriented Benchmarks)

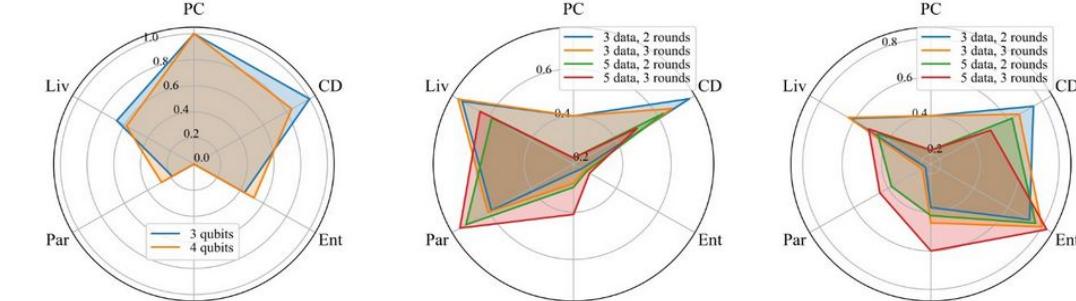
DM_simulator



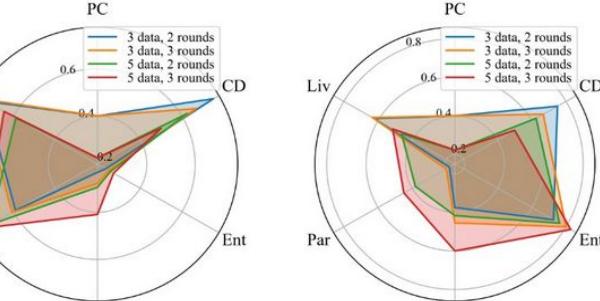
Feature maps and sample circuits for each benchmark (SupermarQ Benchmarking Framework)



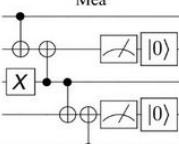
(a) GHZ



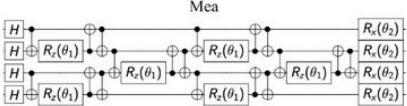
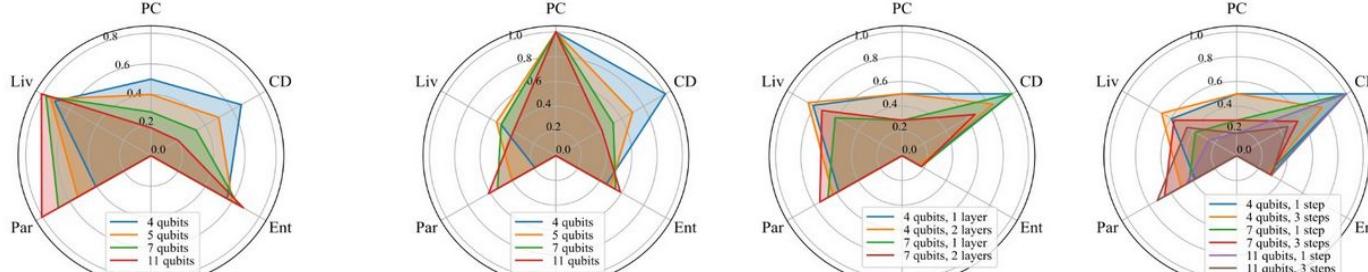
(b) Mermin-Bell



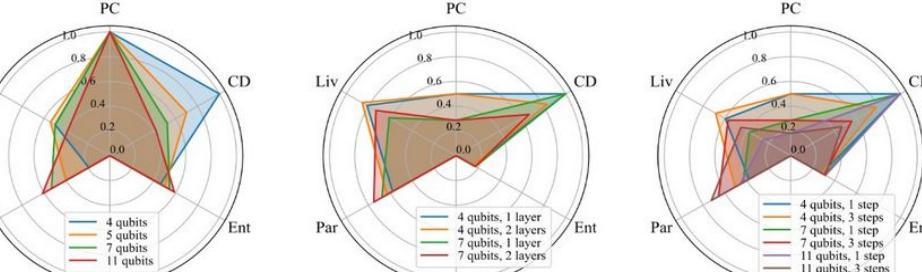
(c) Phase Code



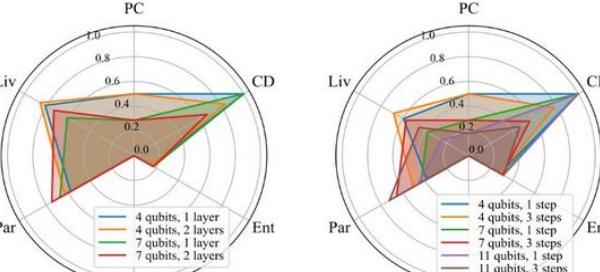
(d) Bit Code



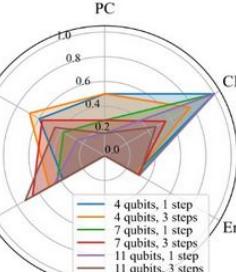
(e) ZZ-SWAP QAOA



(f) Vanilla QAOA



(g) VQE



(h) Hamiltonian Simulation

Figure: Feature maps and sample circuits for each of the benchmarks evaluated

- Program Communication
- Critical-Depth
- Entanglement-Ratio
- Parallelism
- Liveness
- Measurement

Feature Description (SupermarQ)

Program Communication: Captures the level of interaction required between qubits, ranging from simple nearest-neighbor interactions to fully connected communication. This is critical for understanding the impact of qubit connectivity in a quantum device.

Critical Depth: Quantifies the total number of computational steps or layers in a quantum circuit, indicating temporal complexity and the influence of coherence times.

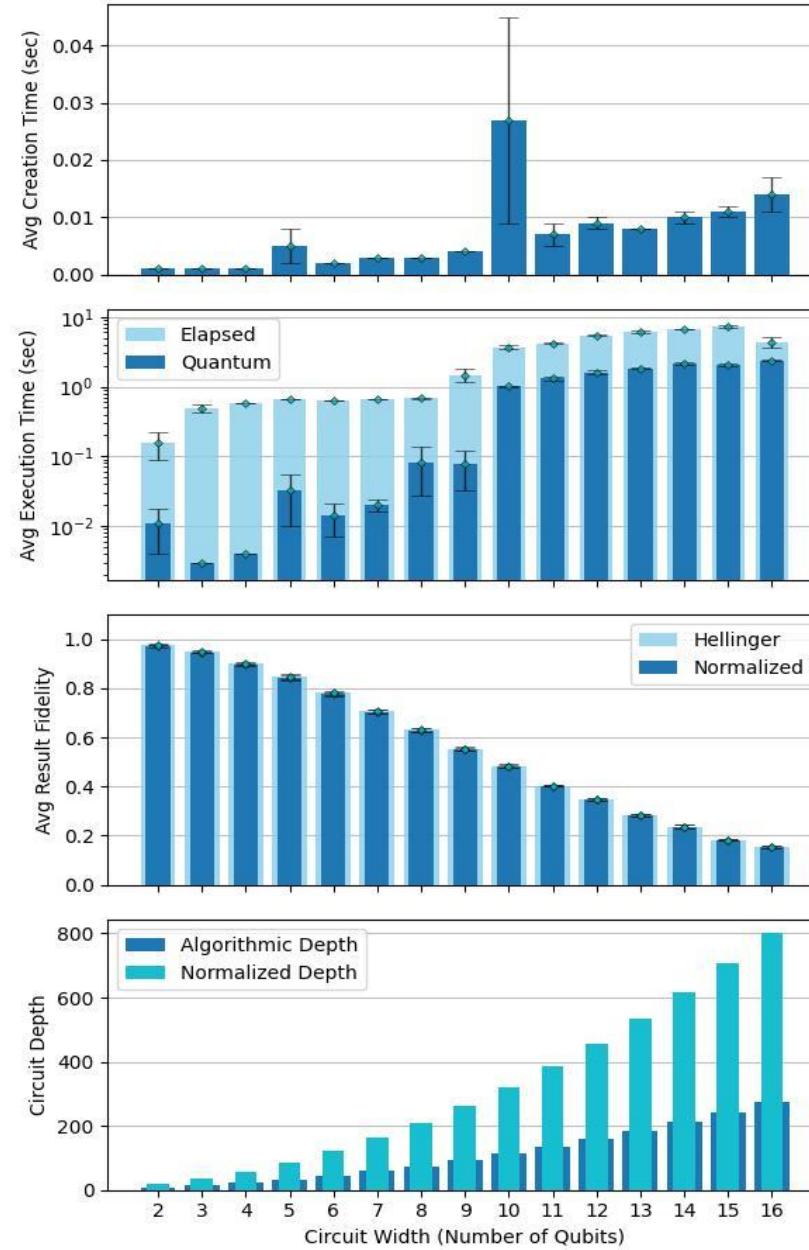
Entanglement-Ratio : quantifies the level of entanglement present in a quantum circuit. It measures the proportion of qubits involved in multi-qubit entanglement during the execution of a quantum program

Parallelism: Highlights the simultaneous execution of quantum gates across different qubits, offering insights into the utilization efficiency of available hardware.

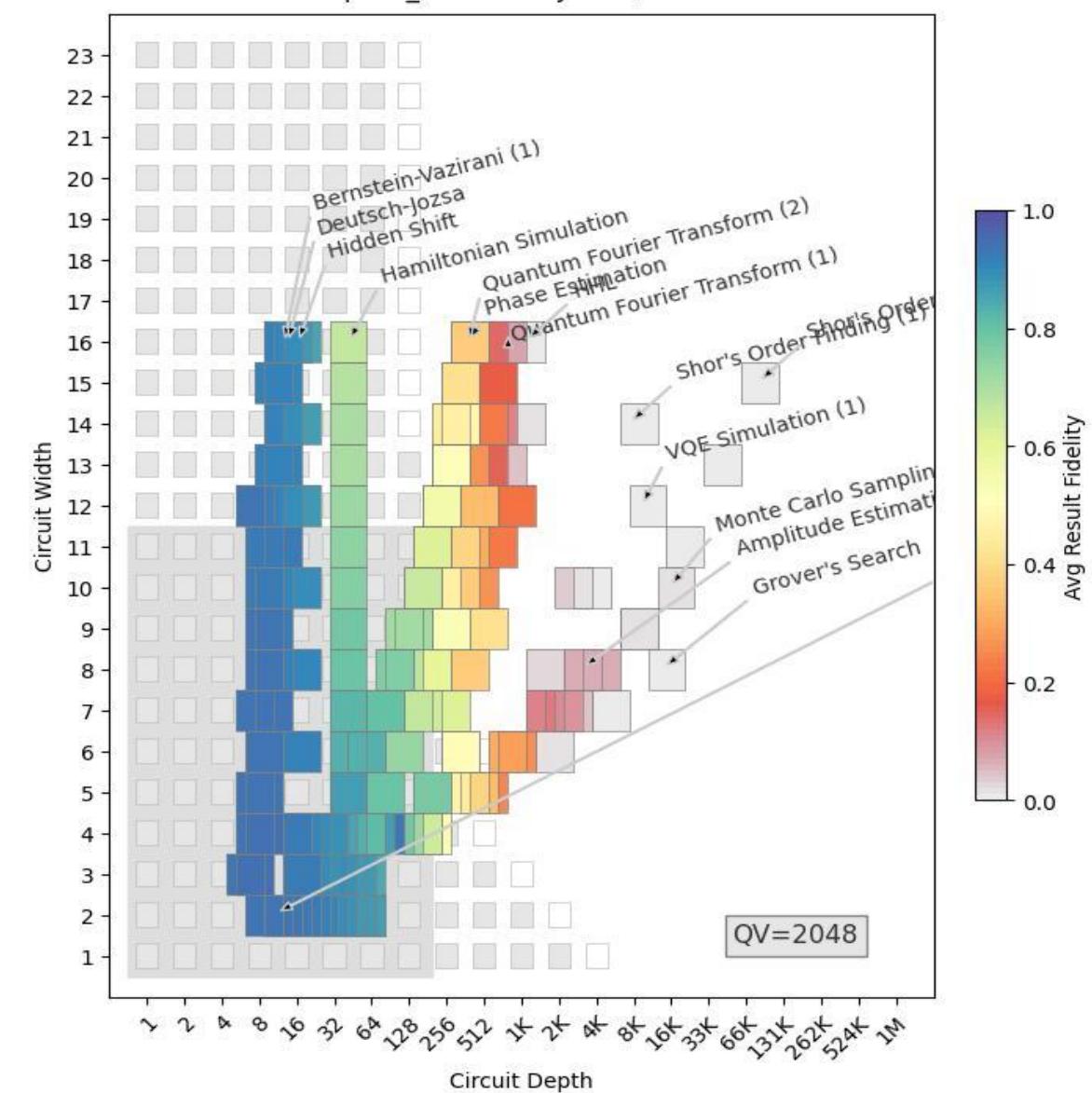
Liveness: Liveness quantifies the fraction of qubits that perform meaningful operations (gates or measurements) as opposed to being idle during a given time frame,It reflects how evenly computational work is distributed across the available qubits

Measurement Density: Indicates how frequently measurements occur, affecting error propagation and overall accuracy.

Benchmark Results - Quantum Fourier Transform (1) - Qiskit
Device=qasm_simulator Jul 09, 2024 08:04:44 UTC



Volumetric Positioning - All Applications (Merged)
Device=qasm_simulator Jul 09, 2024 08:21:59 UTC



Bench marking over GPU A100 for Qasm simulator

<https://tinyurl.com/qsim2024>

qctoolkit.in

<https://tinyurl.com/qsim2024>

qctoolkit.in

[\(qcworkbench@cdac.in\) Email for requesting access](mailto:qcworkbench@cdac.in)

<https://qniverse.in>

<https://qniverse.in/getting-access-qniverse/>