

QuREKA-lab

QuREKA Lab

QuREKA Lab is a **dedicated quantum algorithm development environment** established by creating a Quantum Server.

- **Server-Based Environment:** You can access the lab by clicking the **[Open]** button in the Quantum Server list. It provides a development interface powered by JupyterLab.
- **Customized Specifications:** The development environment is configured precisely according to the hardware specifications (CPU, RAM, GPU, and Storage) you selected during the Quantum Server creation step.
- **Consistent Development Experience:** Based on high-performance computing resources allocated according to your subscription plan, you can reliably perform complex quantum simulations and algorithm designs.

- [Notebook](#)
- [Quantum Composer](#)

Notebook

QuREKA Lab provides a powerful Python development environment based on JupyterLab, supporting an efficient workflow through dedicated extensions and AI-powered tools optimized for quantum algorithm development.

1. Jupyter Notebook Environment

- **Python-Based Development:** Utilize standard Python libraries and quantum computing frameworks within a JupyterLab-based environment.
- **Instant Start:** Create new Jupyter Notebook files and begin development immediately.
- **Sample Files Provided: Sample Notebooks** containing example code for major quantum algorithms are provided by default to assist with learning and practice.

2. Dedicated Extensions and Support Tools

QuREKA Lab places optimized tools in the left and right sidebars for development convenience.

- **AI Assistant (Left Panel):** Provides **AI-powered assistance** for quantum algorithm development and coding. You can easily access the conversational assistant from the left sidebar to receive support for complex implementation processes.

QuREKA File Edit View Run Kernel Git Tabs

Qubenaut 10:21

Hello, I am CUDA-Q Tutor Qubenaut, here to help you understand and effectively use CUDA-Q — from conceptual explanations to example code writing and error analysis. Feel free to ask any questions whenever you need assistance.

You can use the commands below to perform your tasks more conveniently.

- `/clear` — Clear the chat window
- `/export` — Export chat history to a Markdown file
- `/fix` — Fix an error cell selected in your notebook
- `/generate` — Generate a Jupyter notebook from a text prompt
- `/help` — Display this help message

jovyan 10:22

Explain the code below.

```
ort cudaq
from typing import List
```

Qubenaut 10:22

The code imports the `cudaq` library and utilizes the `typing` module to define type hints for lists. Specifically, `from typing import List` allows you to specify that variables of type `List` will hold collections of items. This is good practice for improving code readability and maintainability, as it clarifies the expected data types.

Source 1: https://nvidia.github.io/cuda-quantum/latest/using/examples/hardware_provider
Source 2: <https://blog.langchain.com/semi-structured-multi-modal-rag> Source 3: <https://www.datacamp.com/tutorial/how-to-improve-rag-performance-5-key-techniques-with-examples>

Ask Qubenaut

Untitled.ipynb | cudaq_BV_Is.ipynb

Bernstein-Vazirani Algorithm

The Bernstein-Vazirani Algorithm assumes the existence of a black-box function f defined as:

$$f(x) = s \cdot x \pmod{2}$$

In this context, x consists of n bits, and our goal is to determine the hidden value s . A quantum computer can find the value of s with few operations by amplifying the probability amplitude of the correct answer.

```
[1]: import cudaq
from typing import List
Last executed at 2026-01-21 17:11:25 in 1.75s
```

1. Problem Definition

We define the bits that constitute s .

```
[2]: qubit_count = 5 # Set to around 30 qubits if you have GPU access
secret_string = [1, 1, 0, 1,
                 0] # Change the secret string to whatever you prefer
assert qubit_count == len(secret_string)
Last executed at 2026-01-21 17:11:26 in 5ms
```

2. Quantum Circuit Design

We design a quantum circuit capable of amplifying the correct solution.

```
[3]: @cudaq.kernel
def oracle(register: cudaq.qview, auxiliary_qubit: cudaq.qubit,
           secret_string: List[int]):
    for index, bit in enumerate(secret_string):
        if bit == 1:
            x.ctrl(register[index], auxiliary_qubit)
Last executed at 2026-01-21 17:11:28 in 12ms
```

```
[4]: @cudaq.kernel
def bernstein_vazirani(secret_string: List[int]):
    qubits = cudaq.qvector(len(secret_string)) # register of size n
    auxiliary_qubit = cudaq.qubit() # auxiliary qubit

    # Prepare the auxiliary qubit.
    x(auxiliary_qubit)
    h(auxiliary_qubit)

    # Place the rest of the register in a superposition state.
    h(qubits)

    # Query the oracle.
    oracle(qubits, auxiliary_qubit, secret_string)

    # Apply another set of Hadamards to the register
```

• Right Panel Extensions:

- **Task (Job Monitoring):** Monitor the real-time status and success/failure of your submitted quantum jobs.

Jobs

Search by job name, provider, or ID... All Status ▾

3 jobs total

SUBMITTED / JOB ID	PROVIDER / DEVICE	STATUS	
2026. 02. 05. 오후 07:10 38c10c41-d59...	SDT QubeSIM-MIMIQ	Completed	▾
2026. 02. 05. 오후 07:10 dfd0a2f4-f27...	SDT QubeSIM-MIMIQ	Completed	▴

Execution Info

Job ID	dfd0a2f4-f273-4045-bf1a-9df4cd35f524
Submitted At	2026-02-05 19:10:15.945
Started At	2026-02-05 19:10:16.723
Completed At	2026-02-05 19:10:17.440
Credits Used	-

Job Details

Qubits / Shots	2 / 1,000
Circuit	Copy quantum circuit
Result	Download

2026. 02. 05. 오후 07:10 SDT QubeSIM-MIMIQ Completed ▴

- **Resource (Resource Information):** View a list of available QPUs and simulators along with their detailed specifications.

Resources

Search by provider or resource name All Types ▾

6 resources total

PROVIDER	RESOURCE	TYPE	QUBITS	STATUS	
Rigetti	Ankaa-3	QPU	82	ONLINE	▾
IQM	Emerald	QPU	54	ONLINE	▾
IonQ	Forte-1	QPU	36	ONLINE	▾
IQM	Garnet	QPU	20	ONLINE	▾
SDT	Kreo-SC20	QPU	20	OFFLINE	▴

Backend Name sdt.kreo-sc20

Access Type CUDA-Q

Policy 49.5 credits per job
0.23925 credits per shot

SDT	QubeSIM-MIMIQ	SIMULATOR	100	OFFLINE	▾
-----	---------------	-----------	-----	---------	---

← 1 →

JOB

RESOURCES

WORKSPACE

DEBUGGER

- **Workspace (Workspace Information):** Check current workspace details and your real-time credit balance.

Workspace

R research-lab Team

Subscription **Pro**

Credits **2,480 credits** 

My Role **Admin**

Workspace Details

Members **3 members**

Created **2026-02-05**

Account Info

Account **quireka@sdt.inc**



JOBS



RESOURCES



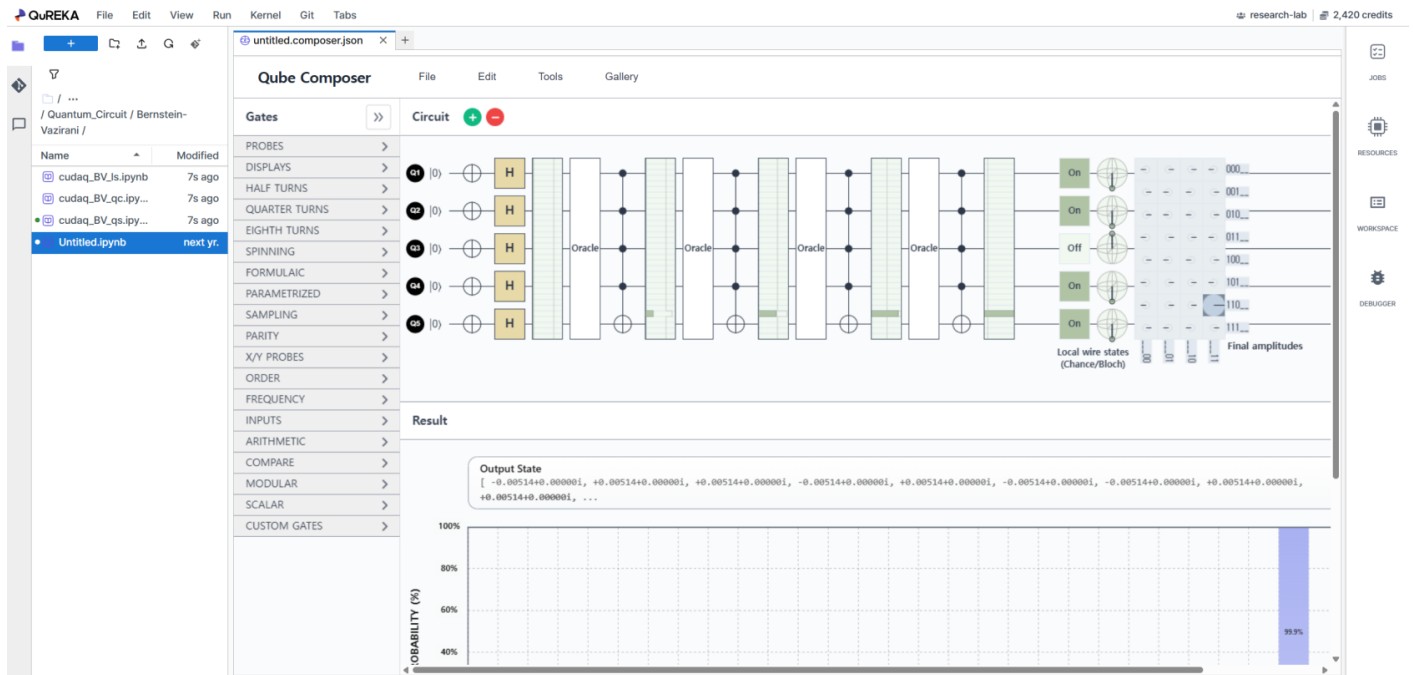
WORKSPACE



DEBUGGER

Quantum Composer

Quantum Composer is a **GUI-based quantum circuit design tool** utilizing drag-and-drop functionality. It allows users to intuitively construct circuits without complex coding and view simulation results in real-time.



Key Features

- **Intuitive Design:** Build circuits by placing gates and efficiently manage logical structures using slicers and barriers.
- **Real-time Simulation:** Instantly updates output states and histograms for up to 16 qubits using GPU acceleration upon any design modification.
- **Visual Analysis:** Provides a clear understanding of qubit phase changes and dynamic state flows through graphics and animations.
- **Multi-Environment Support:** Convert designed circuits into over 10 different quantum languages, such as Qiskit and Cirq, or directly submit them to actual QPU and simulator resources.