

# 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

Qubonaut 10:21

Hello, I am CUDA-Q Tutor Qubonaut, 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
```

Qubonaut 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.html](https://nvidia.github.io/cuda-quantum/latest/using/examples/hardware_provider.html)  
 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 Qubonaut

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.



## 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 →

JOBS

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 **qureka@sdt.inc**



JOBS



RESOURCES



WORKSPACE



DEBUGGER

Revision #4

Created 2025-12-09 04:44:47 UTC by Admin

Updated 2026-02-05 13:29:49 UTC by Admin