Benchmarking gate-based quantum computers via certification of qubit von Neumann measurements

Paulina Lewandowska[1,*] **and Martin Beseda**[1,2]

[1]IT4Innovations, VSB - Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava, Czech Republic
[2]Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, via Vetoio, I-67010 Coppito-L'Aquila, Italy
[*]paulina.lewandowska@vsb.cz

Appendix A: PyQBench as CLI

In this subsection, we will demonstrate how `qbench` package can be used as CLI. Here, we describe how to define an experiment file, a backend file and diffences between YML output files from asynchronous and synchronous experiments.

Defining the experiment

Now we present how to prepare the configuration files. The first configuration file describes the experiment scenario to be executed.

Listing 1. Defining the experiment

```
type: certification-fourier
qubits:
        - target: 0
          ancilla: 1
angles:
        start: 0
        stop: 2 * pi
        num_steps: 8
delta: 0.05
gateset: ibmq
method: direct_sum
num_shots: 10000
```

The experiment file contains the following fields:

- `type`: a string describing the type of experiment. Currently, we command two typos: `discrimination-fourier` and `certification-fourier`.
- `qubits`: a list enumerating pairs of qubits on which the experiment should be run. We describe a particular pair of qubits using `target` and `ancilla` keys to emphasize that the role of qubits in the experiment is distinguished. In our case, we run benchmark on one pair of qubits choosing the `target` as qubit 0, and the `ancilla` a qubit 1.
- `angles`: an object describing the range of angles for the parameterized Fourier family of measurements. The range is always uniform, starts at `start`, ends at `stop` and contains `num_steps` points, including both `start` and `stop`. The `start` and `stop` can be arithmetic expressions using `pi` literal. Our example contains eight points: $\frac{k\pi}{4}$, where $k = 0, \ldots, 8$.
- `delta`: a float defining the statistical significance chosen from the interval $[0, 1]$.
- `gateset`: a string describing the set of gates for decomposing the circuits describing experiment. We can choose `ibmq` corresponding to decompositions compatible with IBM Q devices. Alternatively, one might wish to turn off the decomposition by using a special value `generic`.

- `method`: a string, either `postselection` or `direct_sum` determining which implementation of the conditional measurement is used.
- `num_shots`: an integer defining number of shots are performed in the particular experiment.

The second configuration file describes the backend. Below we describe an example YAML file describing IBM Q backend named Kyiv. Nevertheless, the syntax for each IBM Q backend will be similar. Note that IBM Q backends typically require an access token to IBM Quantum Experience. The token is configured in `QISKIT_IBM_TOKEN` environmental variable. The following listing presents the description of the IBM Q backend for the asynchronous mode.

Listing 2. Defining IBMQ backend

```
name: ibm_kyiv
asynchronous: true
provider:
        hub: ibm-q
        group: open
        project: main
```

YML output files

Below we present the YML output files of synchronous and asynchronous modes.

The result of running the above command can be twofold:

- If the backend is synchronous, the output will contain measurement outcomes (bitstrings) for each of the circuits run.
- If backend is asynchronous, the output will contain intermediate data containing, `job_ids` correlated with the circuit they correspond to.

Synchronous mode

Listing 3. YML output file of synchronous experiment

```
metadata:
  experiments:
    type: certification-fourier
    qubits:
    - {target: 0, ancilla: 1}
    angles: {start: 0.0, stop: 6.283185307179586, num_steps: 8}
    delta: 0.05
    gateset: ibmq
    method: direct_sum
    num_shots: 10000
  backend_description:
    name: ibm_kyiv
    asynchronous: false
    provider: {group: open, hub: ibm-q, project: main}
data:
- target: 0
  ancilla: 1
  phi: 0.0
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'00': 4787, '01': 4663, '11': 314, '10': 236}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
```

```
    mitigated_histogram: {'10': 0.02272105079666005,
            '11': 0.0308324019917434, '01': 0.4686639719002791,
            '00': 0.47778257531131735}
- target: 0
  ancilla: 1
  phi: 0.8975979010256552
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'01': 3158, '11': 1841, '10': 2121, '00': 2880}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'11': 0.18503494944149518,
            '10': 0.2119853847642628, '00': 0.2863022864686138,
            '01': 0.3166773793256281}
- target: 0
  ancilla: 1
  phi: 1.7951958020513104
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'11': 3946, '10': 4107, '01': 933, '00': 1014}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'01': 0.09187983560151308,
            '00': 0.0992818314007015, '11': 0.3977454665741516,
            '10': 0.4110928664236337}
- target: 0
  ancilla: 1
  phi: 2.6927937030769655
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'10': 4905, '11': 4844, '00': 121, '01': 130}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'00': 0.00971145661283892,
            '01': 0.0107234135301044, '11': 0.4884707846971315,
            '10': 0.491094345159925}
- target: 0
  ancilla: 1
  phi: 3.5903916041026207
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'11': 4727, '10': 5047, '01': 119, '00': 107}
```

```
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'00': 0.008243326178401936,
            '01': 0.0096749343410262, '11': 0.4766264355219874,
            '10': 0.5054553039585844}
- target: 0
  ancilla: 1
  phi: 4.487989505128276
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'10': 4067, '00': 924, '01': 1045, '11': 3964}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'00': 0.09020755763218545,
            '01': 0.103168725838722, '11': 0.39955085514435557,
            '10': 0.4070728613847364}
- target: 0
  ancilla: 1
  phi: 5.385587406153931
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'10': 1973, '11': 1989, '00': 3069, '01': 2969}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'10': 0.1969715269854367,
            '11': 0.2000488072203213, '01': 0.2975337874613869,
            '00': 0.305445878332855}
- target: 0
  ancilla: 1
  phi: 6.283185307179586
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram: {'00': 4832, '01': 4661, '11': 257, '10': 250}
    mitigation_info:
      target: {prob_meas0_prep1: 0.00539999999999996,
            prob_meas1_prep0: 0.0018}
      ancilla: {prob_meas0_prep1: 0.0048000000000000265,
            prob_meas1_prep0: 0.0018}
    mitigated_histogram: {'10': 0.024153348441373016,
            '11': 0.0250715357945843, '01': 0.4684820500233045,
            '00': 0.48229306574073816}
```

The data whereas includes `target, ancilla, phi, delta` and `results_per_circuit` information. The first four pieces of information have already been described. The last data `results_per_circuit` gives us the following

additional information:

- `histogram`: the dictionary with measurements' outcomes. The keys represent possible bitstrings, whereas the values are the number of occurrences.
- `mitigation_info`: the parameters `prob_meas0_prep1` and `prob_meas1_prep0` contains information that are used for error mitigation using the MThree method[1] and can be found using `backends.properties().qubits`. If this information is available, it will be stored in the mitigation info field; otherwise, this field will be absent.
- `mitigation_histogram`: the histogram with probabilities of measurements' outcomes after the error mitigation.

Asynchronous mode

Listing 4. YML output file of asynchronous mode with intermediate data

```
metadata:
  experiments:
    type: certification-fourier
    qubits:
    - {target: 0, ancilla: 1}
    angles: {start: 0.0, stop: 6.283185307179586, num_steps: 8}
    delta: 0.05
    gateset: ibmq
    method: direct_sum
    num_shots: 10000
  backend_description:
    name: ibm_kyiv
    asynchronous: true
    provider: {group: open, hub: ibm-q, project: main}
data:
- job_id: cy60ptkcw2k0008jwsag
  keys:
  - [0, 1, u, 0.0, 0.05]
  - [0, 1, u, 0.8975979010256552, 0.05]
  - [0, 1, u, 1.7951958020513104, 0.05]
  - [0, 1, u, 2.6927937030769655, 0.05]
  - [0, 1, u, 3.5903916041026207, 0.05]
  - [0, 1, u, 4.487989505128276, 0.05]
  - [0, 1, u, 5.385587406153931, 0.05]
  - [0, 1, u, 6.283185307179586, 0.05]
```

Listing 5. YML output file of resolved experiment

```
metadata:
  experiments:
    type: certification-fourier
    qubits:
    - target: 0
      ancilla: 1
    angles:
      start: 0.0
      stop: 6.283185307179586
      num_steps: 8
    delta: 0.05
    gateset: ibmq
    method: direct_sum
    num_shots: 10000
  backend_description:
```

```
    name: ibm_kyiv
    asynchronous: true
    provider:
      group: open
      hub: ibm-q
      project: main
data:
- target: 0
  ancilla: 1
  phi: 0.0
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram:
      '00': 4898
      '01': 4582
      '11': 311
      '10': 209
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '10': 0.01998449638687051
      '11': 0.030549024853314708
      '01': 0.4604864304246869
      '00': 0.4889800483351278
- target: 0
  ancilla: 1
  phi: 0.8975979010256552
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram:
      '11': 1952
      '00': 2876
      '10': 2074
      '01': 3098
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '11': 0.19626874194826874
      '10': 0.2071941128936882
      '00': 0.28595657203781866
      '01': 0.3105805731202244
- target: 0
  ancilla: 1
```

```
    phi: 1.7951958020513104
    delta: 0.05
    results_per_circuit:
    - name: u
      histogram:
        '01': 905
        '11': 4022
        '10': 4085
        '00': 988
      mitigation_info:
        target:
          prob_meas0_prep1: 0.00539999999999996
          prob_meas1_prep0: 0.0018
        ancilla:
          prob_meas0_prep1: 0.0048000000000000265
          prob_meas1_prep0: 0.0018
      mitigated_histogram:
        '01': 0.08902728596975329
        '00': 0.09669850424566845
        '11': 0.4054328268424948
        '10': 0.40884138294208333
  - target: 0
    ancilla: 1
    phi: 2.6927937030769655
    delta: 0.05
    results_per_circuit:
    - name: u
      histogram:
        '10': 4990
        '11': 4825
        '00': 81
        '01': 104
      mitigation_info:
        target:
          prob_meas0_prep1: 0.00539999999999996
          prob_meas1_prep0: 0.0018
        ancilla:
          prob_meas0_prep1: 0.0048000000000000265
          prob_meas1_prep0: 0.0018
      mitigated_histogram:
        '00': 0.005669915398114243
        '01': 0.008121105338749025
        '11': 0.4865404579166901
        '10': 0.4996685213464466
  - target: 0
    ancilla: 1
    phi: 3.5903916041026207
    delta: 0.05
    results_per_circuit:
    - name: u
      histogram:
        '11': 4826
        '10': 4964
        '00': 103
        '01': 107
```

```
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '00': 0.007888851269304064
      '01': 0.008418779091074404
      '11': 0.4866456850507466
      '10': 0.49704668458887485
- target: 0
  ancilla: 1
  phi: 4.487989505128276
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram:
      '11': 3954
      '10': 4124
      '00': 889
      '01': 1033
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '00': 0.08667377941797003
      '01': 0.10197127796072936
      '11': 0.398532348147248
      '10': 0.41282259447405256
- target: 0
  ancilla: 1
  phi: 5.385587406153931
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram:
      '00': 3129
      '10': 1964
      '01': 2977
      '11': 1930
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '11': 0.19408825484296274
```

```
        '10': 0.19608690118683395
        '01': 0.2983573535373757
        '00': 0.3114674904328276
- target: 0
  ancilla: 1
  phi: 6.283185307179586
  delta: 0.05
  results_per_circuit:
  - name: u
    histogram:
      '01': 4768
      '00': 4696
      '10': 241
      '11': 295
    mitigation_info:
      target:
        prob_meas0_prep1: 0.00539999999999996
        prob_meas1_prep0: 0.0018
      ancilla:
        prob_meas0_prep1: 0.0048000000000000265
        prob_meas1_prep0: 0.0018
    mitigated_histogram:
      '10': 0.02325138419612959
      '11': 0.02889276720310536
      '00': 0.4685898728546358
      '01': 0.4792659757461291
```


Appendix B: PyQBench as Python library

In this appendix, we will demonstrate how `qbench` package can be used with user–defined measurement. When used as a library, PyQBench allows the manipulation of certification scheme by, e.g. adding a noise model. The user then defines a unitary matrix U that describes the measurement to be certified, the optimal initial state $|\psi_0\rangle$ and unitaries V_0 and V_1 to create the final measurement. The PyQBench library provides then the following functionalities: assembling circuits for certification scheme, running the whole circuits defining benchmark on specified IBM Q backend and, finally, interpreting the obtained outputs by computing the minimized probability of type II error. Below we present the usage example which can be found in[2].

We will present the certification of the measurement performed in the Hadamard basis to show the usage of PyQBench as a Python library. The explicit formula for initial state in this case reads

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{1}$$

with the final measurements of the forms

$$V_0 = \begin{pmatrix} \sqrt{1-\delta} & \sqrt{\delta} \\ -\sqrt{\delta} & \sqrt{1-\delta} \end{pmatrix}, \tag{2}$$

and

$$V_1 = \begin{pmatrix} -\sqrt{\delta} & \sqrt{1-\delta} \\ \sqrt{1-\delta} & \sqrt{\delta} \end{pmatrix}, \tag{3}$$

for some choice of the statistical significance $\delta \in [0,1]$. Thus, the minimized probability of type II error is equal to

$$p_{\mathrm{II}} = \frac{1}{2}(\sqrt{1-\delta} - \sqrt{\delta})^2. \tag{4}$$

In our example, let assume $\delta = 0.05$. To use the above benchmarking scheme in PyQBench, we first need to construct circuits that can be executed by actual hardware. The circuit taking $|00\rangle$ to the Bell state $|\psi_0\rangle$ comprises the Hadamard gate followed by

CX gate on both qubits. For $V_0^\dagger$ and $V_1^\dagger$ observe that $V_0^\dagger = \mathrm{RY}\left(2\arcsin\left(\sqrt{0.05}\right)\right)$ and $V_1^\dagger = \mathrm{X}\cdot\mathrm{RY}\left(2\arcsin\left(\sqrt{0.05}\right)\right)$ where $\mathrm{RY}(\theta)$ is rotation gate around the Y axis defined by

$$\mathrm{RY}(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}. \tag{5}$$

We will now demonstrate how to implement this theoretical scheme in PyQBench with some modifications. For this example we will use the Qiskit Aer simulator[3]. First, we import the necessary functions and classes from PyQBench and Qiskit.

Listing 6. Importing the necessary functions and classes

```
from qiskit_aer import StatevectorSimulator
from qiskit import QuantumCircuit
import numpy as np
from qbench.schemes.postselection import
    benchmark_certification_using_postselection
from qbench.schemes.direct_sum import
    benchmark_certification_using_direct_sum
```

To implement the certification scheme in PyQBench, we need to define all the necessary components as Qiskit instructions. To do so, we first define `QuantumCircuit(2)` acting on qubit 0 and 1 and then use the `instruction()` method.

Listing 7. Defining components for Hadamard experiment

```
def state_prep():
    circuit = QuantumCircuit(2)
    circuit.h(0)
    circuit.cx(0,1)
    return circuit.to_instruction()

def u_dag():
    circuit = QuantumCircuit(1)
    circuit.h(0)
    return circuit.to_instruction()

def v0_dag():
    circuit = QuantumCircuit(1)
    circuit.ry(2 * np.arcsin(np.sqrt(0.05)), 0)
    return circuit.to_instruction()

def v1_dag():
    circuit = QuantumCircuit(1)
    circuit.ry(2 * np.arcsin(np.sqrt(0.05)), 0)
    circuit.x(0)
    return circuit.to_instruction()

def v0_v1_direct_sum_dag():
    circuit = QuantumCircuit(2)
    circuit.p(-np.pi, 0)
    circuit.ry(-2 * np.arcsin(np.sqrt(0.05)), 0)
    circuit.cx(0, 1)
    return circuit.to_instruction()
```

We also need to construct a backend object, which is an instance Aer simulator.

Listing 8. Defining a backend

```
simulator = StatevectorSimulator()
```

Now, when one wishes to run the experiment without any modifications on a given backend, it is enough to run `benchmark_certification_using_postselection` or `benchmark_certification_using_direct_sum` function, depending on the user preference.

Listing 9. Simulation benchmark by using postselection

```
postselection_result =
    benchmark_certification_using_postselection(
    backend=simulator,
    target=0,
    ancilla=1,
    state_preparation=state_prep(),
    u_dag=u_dag(),
    v0_dag=v0_dag(),
    v1_dag=v1_dag(),
    num_shots_per_measurement=10000)
```

Listing 10. Simulation benchmark by using direct sum

```
direct_sum_result = benchmark_certification_using_direct_sum(
    backend=simulator,
    target=0,
    ancilla=1,
    state_preparation=state_prep(),
    u_dag=u_dag(),
    v0_v1_direct_sum_dag=v0_v1_direct_sum_dag(),
    num_shots_per_measurement=10000)
```

The `postselection_result` and `direct_sum_result` variables contain now the empirical probabilities of type II error. We can compare them with the theoretical value and compute the absolute error.

Listing 11. Examining the benchmark results

```
p_succ = (1/np.sqrt(2) * np.sqrt(0.95) +
        - 1/np.sqrt(2) * np.sqrt(0.05))**2
print(f"Analytical p_succ = {p_succ}")
print(f"Postselection: p_succ = {postselection_result},
abs_error ={np.abs(p_succ - postselection_result)}")
print(f"Direct_sum: p_succ = {direct_sum_result},
abs_error ={np.abs(p_succ - direct_sum_result)}")
```

```
Analytical p_succ = 0.2820550528229661
Postselection: p_succ = 0.28322830780328,
    abs_error = 0.001173254980313898
Direct_sum: p_succ = 0.28333, abs_error = 0.0012749471770339138
```

But what if we want to modify this process? PyQBench provides functions performing: assembly of circuits needed for experiment under providing the components and interpretation of the obtained measurements.

For the rest of this example, we focus only on the postselection case, as the direct sum case is analogous. We continue by importing two more functions from PyQBench.

Listing 12. Assembling circuits

```
from qbench.schemes.postselection import (
    assemble_circuits_certification_postselection,
    compute_probabilities_certification_postselection)

circuits = assemble_circuits_certification_postselection(
    target=0,
```

```
    ancilla=1,
    state_preparation=state_prep(),
    u_dag=u_dag(),
    v0_dag=v0_dag(),
    v1_dag=v1_dag())
```

The function `assemble_circuits_certification_postselection` creates two circuits and places them in a dictionary with keys "`u_v0`", "`u_v1`". Now we will run our circuits using noisy and noiseless simulation. We start by creating a noise model using Qiskit.

Listing 13. Adding noise model

```
from qiskit_aer.noise import NoiseModel, ReadoutError

error = ReadoutError([[0.75, 0.25], [0.8, 0.2]])

noise_model = NoiseModel()

noise_model.add_readout_error(error, [0])
noise_model.add_readout_error(error, [1])
```

Now we can execute the circuits with and without noise. To do this, we will use Qiskit's `run` function. It should be mentioned that we have to keep track of which measurements correspond to which circuit. We do so by fixing an ordering on the keys in the circuits dictionary.

Listing 14. Running circuits

```
keys_ordering = ["u_v0", "u_v1"]

all_circuits = [circuits[key] for key in keys_ordering]

counts_noisy = simulator.run(
    all_circuits,
    backend=simulator,
    noise_model=noise_model,
    shots=100000).result().get_counts()

counts_noiseless = simulator.run(
    all_circuits,
    backend=simulator,
    shots=100000).result().get_counts()
```

Finally, we use `compute_probabilities_certification_postselection` function to compute discrimination probabilities.

Listing 15. Computing probabilities

```
prob_succ_noiseless =
compute_probabilities_certification_postselection(
    u_v0_counts=counts_noiseless[0],
    u_v1_counts=counts_noiseless[1],)

prob_succ_noisy =
compute_probabilities_certification_postselection(
    u_v0_counts=counts_noisy[0],
    u_v1_counts=counts_noisy[1],)
```

We can now examine the results. From the experiment, we obtain `prob_succ_noiseless` = 0.28072503092454 and `prob_succ_noisy` = 0.77564942534484. As expected, for noisy simulations, the result lies further away from the target value of 0.2820550528229661. This concludes our example.

References

1. Mthree package documentation. https://qiskit.org/documentation/partners/mthree/stubs/mthree.M3Mitigation.html. Accessed on 2025-01-01.
2. Hadamard certification experiment. https://github.com/iitis/PyQBench/blob/master/Hadamard_certification_experiment.ipynb. Accessed on 2025-07-01.
3. Qiskit Aer GitHub Repository. https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.StatevectorSimulator.html. Accessed on 2025-01-01.