

PyQBench: a Python library for benchmarking gate-based quantum computers

Konrad Jałowiecki*, Paulina Lewandowska, Łukasz Paweł

*Institute of Theoretical and Applied Informatics, Polish Academy of Sciences,
Baltycka 5, 44-100 Gliwice, Poland*

Abstract

We introduce PyQBench, an innovative open-source framework for benchmarking gate-based quantum computers. PyQBench can benchmark NISQ devices by verifying their capability of discriminating between two von Neumann measurements. PyQBench offers a simplified, ready-to-use, command line interface (CLI) for running benchmarks using a predefined parametrized Fourier family of measurements. For more advanced scenarios, PyQBench offers a way of employing user-defined measurements instead of predefined ones.

Keywords: Quantum computing, Benchmarking quantum computers, Discrimination of quantum measurements, Discrimination of von Neumann measurements, Open-source, Python programming

PACS: 03.67.-a, 03.67.Lx

2000 MSC: 81P68

*Corresponding author

Email address: `dexter2206@gmail.com` (Konrad Jałowiecki)

Current code version

C1	Current code version	0.1.1
C2	Permanent link to code/repository used for this code version	https://github.com/iitis/PyQBench
C3	Code Ocean compute capsule	https://codeocean.com/capsule/89088992-9a27-4712-8525-d92a9b23060f/tree
C4	Legal Code License	Apache License 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, Qiskit, AWS Braket
C7	Compilation requirements, operating environments & dependencies	<code>Python >= 3.8</code> <code>numpy ~= 1.22.0</code> <code>scipy ~= 1.7.0</code> <code>pandas ~= 1.5.0</code> <code>amazon-braket-sdk >= 1.11.1</code> <code>pydantic ~= 1.9.1</code> <code>qiskit ~= 0.37.2</code> <code>mthree ~= 1.1.0</code> <code>tqdm ~= 4.64.1</code> <code>pyyaml ~= 6.0</code> <code>qiskit-braket-provider ~= 0.0.3</code>
C8	If available Link to developer documentation/manual	https://pyqbench.readthedocs.io/en/latest/
C9	Support email for questions	dexter2206@gmail.com

Table 1: Code metadata

1. Motivation and significance

Noisy Intermediate-Scale Quantum (NISQ) [1] devices are storming the market, with a wide selection of devices based on different architectures and accompanying software solutions. Among hardware providers offering public access to their gate-based devices, one could mention Rigetti [2], IBM [3], Oxford Quantum Group [4], IonQ [5] or Xanadu [6]. Other vendors offer devices operating in different paradigms. Notably, one could mention D-Wave [7] and their quantum annealers, or QuEra devices [8] based on neutral atoms. Most vendors provide their own software stack and application programming interface for accessing their devices. To name a few, Rigetti’s computers are available through their Forest SDK [9] and PyQuil library [10] and IBM Q [3]

12 computers can be accessed through Qiskit [11] or IBM Quantum Experience
13 web interface [12]. Some cloud services, like Amazon Braket [13], offer ac-
14 cess to several quantum devices under a unified API. On top of that, several
15 libraries and frameworks can integrate with multiple hardware vendors. Ex-
16 amples of such frameworks include IBM Q’s Qiskit or Zapata Computing’s
17 Orquestra [14].

18 It is well known that NISQ devices have their limitations [15]. The ques-
19 tion is to what extent those devices can perform meaningful computations?
20 To answer this question, one has to devise a methodology for benchmarking
21 them. For gate-based computers, on which this paper focuses, there al-
22 ready exist several approaches. One could mention randomized benchmark-
23 ing [16, 17, 18, 19, 20], benchmarks based on the quantum volume [21, 22, 23].

24 In this paper, we introduce a different approach to benchmarking gate-
25 based devices with a simple operational interpretation. In our method, we
26 test how well the given device is at guessing which of the two known von
27 Neumann measurements were performed during the experiment. We imple-
28 mented our approach in an open-source Python library called PyQBench.
29 The library supports any device available through the Qiskit library, and
30 thus can be used with providers such as IBM Q or Amazon Braket. Along
31 with the library, the PyQBench package contains a command line tool for
32 running most common benchmarking scenarios.

33 2. Existing benchmarking methodologies and software

34 Unsurprisingly, PyQBench is not the only software package for bench-
35 marking gate-based devices. While we believe that our approach has signif-
36 icant benefits over other benchmarking techniques, for completeness, in this
37 section we discuss some of the currently available similar software.

38 Probably the simplest benchmarking method one could devise is simply
39 running known algorithms and comparing outputs with the expected ones.
40 Analyzing the frequency of the correct outputs, or the deviation between
41 actual and expected outputs distribution provides then a metric of the per-
42 formance of a given device. Libraries such as Munich Quantum Toolkit
43 (MQT) [24, 25] or SupermarQ [26, 27] contain benchmarks leveraging mul-
44 tiple algorithms, such as Shor’s algorithm or Grover’s algorithm. Despite
45 being intuitive and easily interpretable, such benchmarks may have some
46 problems. Most importantly, they assess the usefulness of a quantum device
47 only for a very particular algorithm, and it might be hard to extrapolate
48 their results to other algorithms and applications. For instance, the inability
49 of a device to consistently find factorizations using Shor’s algorithms does
50 not tell anything about its usefulness in Variational Quantum Algorithm’s.

Another possible approach to benchmarking quantum computers is randomized benchmarking. In this approach, one samples circuits to be run from some predefined set of gates (e.g. from the Clifford group) and tests how much the output distribution obtained from the device running these circuits differs from the ideal one. It is also common to concatenate randomly chosen circuits with their inverses (which should yield the identity circuit) and run those concatenated circuits on the device. Libraries implementing this approach include Qiskit [28] or PyQuil [29].

Another quantity used for benchmarking NISQ devices is quantum volume. The quantum volume characterizes capacity of a device for solving computational problems. It takes into account multiple factors like number of qubits, connectivity and measurement errors. The Qiskit library allows one to measure quantum volume of a device by using its `qiskit.ignis.verification.quantum_volume`. Other implementations of Quantum Volume can be found as well, see e.g. [30].

3. Preliminaries and discrimination scheme approach

In this section we describe how the benchmarking process in PyQBench works. To do so, we first discuss necessary mathematical preliminaries. Then, we present the general form of the discrimination scheme used in PyQBench and practical considerations on how to implement it taking into account limitations of the current NISQ devices.

3.1. Mathematical preliminaries

Let us first recall the definition of a von Neumann measurement, which is the only type of measurement used in PyQBench. A von Neumann measurement \mathcal{P} is a collection of rank-one projectors $\{|u_0\rangle\langle u_0|, \dots, |u_{d-1}\rangle\langle u_{d-1}|\}$, called effects, that sum up to identity, i.e. $\sum_{i=0}^{d-1} |u_i\rangle\langle u_i| = \mathbb{1}$. If U is a unitary matrix of size d , one can construct a von Neumann measurement \mathcal{P}_U by taking projectors onto its columns. In this case we say that \mathcal{P}_U is described by the matrix U .

Typically, NISQ devices can only perform measurements in computational Z -basis, i.e. $U = \mathbb{1}$. To implement an arbitrary von Neumann measurement \mathcal{P}_U , one has to first apply U^\dagger to the measured system and then follow with Z -basis measurement. This process, depicted in Fig. 1, can be viewed as performing a change of basis in which measurement is performed prior to measurement in the computational basis.

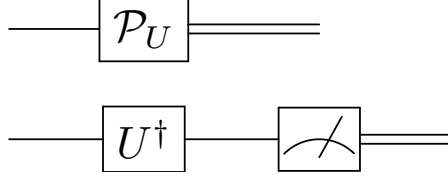


Figure 1: Implementation of a von Neumann measurement using measurement in computational basis. The upper circuit shows a symbolic representation of a von Neumann measurement \mathcal{P}_U . The bottom, equivalent circuit depicts its decomposition into a change of basis followed by measurement in the Z basis.

3.2. Discrimination scheme

Benchmarks in PyQBench work by experimentally determining the probability of correct discrimination between two von Neumann measurements by the device under test and comparing the result with the ideal, theoretical predictions.

Without loss of generality¹, we consider discrimination task between single qubit measurements \mathcal{P}_1 , performed in the computational Z -basis, and an alternative measurement \mathcal{P}_U performed in the basis U . Note, however, that the discrimination scheme described below can work regardless of dimensionality of the system, see [31] for details.

In general, the discrimination scheme presented in Fig. 2, requires an auxiliary qubit. First, the joint system is prepared in some state $|\psi_0\rangle$. Then, one of the measurements, either \mathcal{P}_U or \mathcal{P}_1 , is performed on the first part of the system. Based on its outcome i , we choose another POVM \mathcal{P}_{V_i} and perform it on the second qubit, obtaining the output in j . Finally, if $j = 0$, we say that the performed measurement is \mathcal{P}_U , otherwise we say that it was \mathcal{P}_1 . Naturally, we need to repeat the same procedure multiple times for both measurements to obtain a reliable estimate of the underlying probability distribution. In PyQBench, we assume that the experiment is repeated the same number of times for both \mathcal{P}_U and \mathcal{P}_1 .

Unsurprisingly, both the $|\psi_0\rangle$ and the final measurements \mathcal{P}_{V_i} have to be chosen specifically for given U to maximize the probability of a correct guess. The detailed description how these choices are made in [32], and for now we will focus only how this scheme can be implemented on the actual devices, assuming that all the components are known.

¹Explaining why we can consider only discrimination scheme between \mathcal{P}_1 and \mathcal{P}_U is beyond the scope of this paper. See [31] for a in depth explanation.

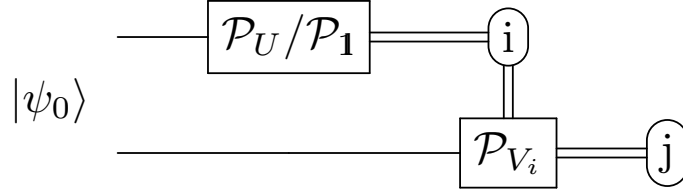


Figure 2: Theoretical scheme of discrimination between von Neumann measurements \mathcal{P}_U and \mathcal{P}_1 .

111 3.2.1. Implementation of discrimination scheme on actual NISQ devices

112 Current NISQ devices are unable to perform conditional measurements,
 113 which is the biggest obstacle to implementing our scheme on real hardware.
 114 However, we circumvent this problem by slightly adjusting our scheme so
 115 that it only uses components available on current devices. For this purpose,
 116 we use two possible options: using a postselection or a direct sum $V_0^\dagger \oplus V_1^\dagger$.

117 **Scheme 1.** (Postselection)

118 The first idea uses a postselection scheme. In the original scheme, we
 119 measure the first qubit and only then determine which measurement should
 120 be performed on the second one. Instead of doing this choice, we can run two
 121 circuits, one with \mathcal{P}_{V_0} and one with \mathcal{P}_{V_1} and measure both qubits. We then
 122 discard the results of the circuit for which label i does not match measurement
 123 label k . Hence, the circuit for postselection looks as depicted in Fig. 3.

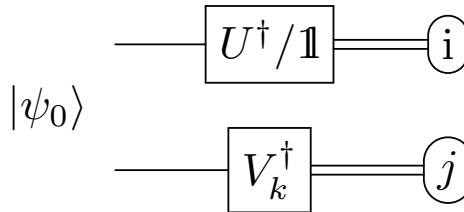


Figure 3: A schematic representation of the setup for distinguishing measurements \mathcal{P}_U and \mathcal{P}_1 using postselection approach. In postselection scheme, one runs such circuits for both $k = 0, 1$ and discards results for cases when there is a mismatch between k and i .

124 To perform the benchmark, one needs to run multiple copies of the post-
 125 selection circuit, with both \mathcal{P}_U and \mathcal{P}_1 . Each circuit has to be run in both
 126 variants, one with final measurement \mathcal{P}_{V_0} and the second with the final mea-
 127 surement \mathcal{P}_{V_1} . The experiments can thus be grouped into classes identified by
 128 tuples of the form (\mathcal{Q}, k, i, j) , where $\mathcal{Q} \in \{\mathcal{P}_U, \mathcal{P}_1\}$ denotes the chosen mea-
 129 surement, $k \in \{0, 1\}$ designates the final measurement used, and $i \in \{0, 1\}$
 130 and $j \in \{0, 1\}$ being the labels of outcomes as presented in Fig. 3. We

131 then discard all the experiments for which $i \neq k$. The total number of valid
 132 experiments is thus:

$$N_{\text{total}} = \#\{(\mathcal{Q}, k, i, j) : k = i\}. \quad (1)$$

133 Finally, we count the valid experiments resulting in successful discrimi-
 134 nation. If we have chosen \mathcal{P}_U , then we guess correctly iff $j = 0$. Similarly,
 135 for \mathcal{P}_1 , we guess correctly iff $j = 1$. If we define

$$N_{\mathcal{P}_U} = \#\{(\mathcal{Q}, k, i, j) : \mathcal{Q} = \mathcal{P}_U, k = i, j = 0\}, \quad (2)$$

$$N_{\mathcal{P}_1} = \#\{(\mathcal{Q}, k, i, j) : \mathcal{Q} = \mathcal{P}_1, k = i, j = 1\}, \quad (3)$$

136 then the empirical success probability can be computed as

$$p_{\text{succ}}(\mathcal{P}_U, \mathcal{P}_1) = \frac{N_{\mathcal{P}_U} + N_{\mathcal{P}_1}}{N_{\text{total}}}. \quad (4)$$

137 The p_{succ} is the quantity reported to the user as the result of the benchmark.

138 **Scheme 2.** (Direct sum)

139 The second idea uses the direct sum $V_0^\dagger \oplus V_1^\dagger$ implementation. Here,
 140 instead of performing a conditional measurement \mathcal{P}_{V_k} , where $k \in \{0, 1\}$, we
 141 run circuits presented in Fig. 4.

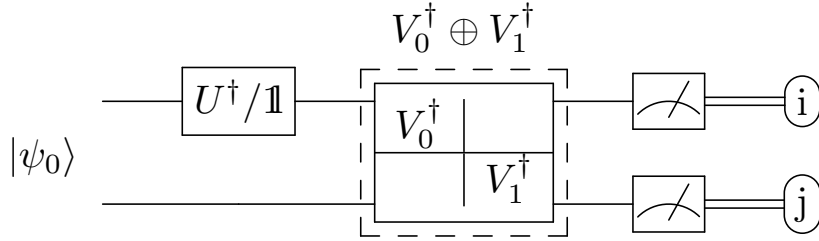


Figure 4: A schematic representation of the setup for distinguishing measurements \mathcal{P}_U and \mathcal{P}_1 using the $V_0^\dagger \oplus V_1^\dagger$ direct sum.

142 One can see why such a circuit is equivalent to the original discrimination
 143 scheme. If we rewrite the block-diagonal matrix $V_0^\dagger \oplus V_1^\dagger$ as follows:

$$V_0^\dagger \oplus V_1^\dagger = |0\rangle\langle 0| \otimes V_0^\dagger + |1\rangle\langle 1| \otimes V_1^\dagger, \quad (5)$$

144 we can see that the direct sum in Eq. (5) commutes with the measurement
 145 on the first qubit. Thanks to this, we can switch the order of operations to
 146 obtain the circuit from Fig. 5. Now, depending on the outcome i , one of the

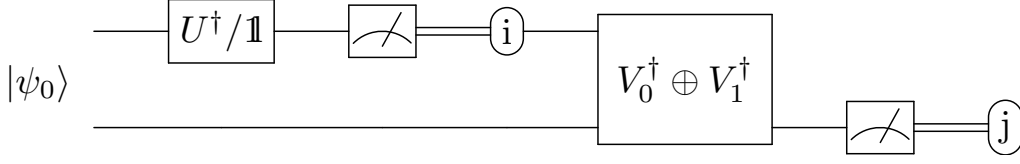


Figure 5: Rewritten representation of the setup for distinguishing measurements \mathcal{P}_U and \mathcal{P}_1 using the $V_0^\dagger \oplus V_1^\dagger$ direct sum.

summands in Eq. (5) vanishes, and we end up performing exactly the same operations as in the original scheme.

In this scheme, the experiment can be characterized by a pair (\mathcal{Q}, i, j) , where $\mathcal{Q} = \{\mathcal{P}_U, \mathcal{P}_1\}$ and $i, j \in \{0, 1\}$ are the output labels. The number of successful trials for U and $\mathbb{1}$, respectively, can be written as

$$N_{\mathcal{P}_U} = \#\{(\mathcal{Q}, i, j) : \mathcal{Q} = \mathcal{P}_U, j = 0\}, \quad (6)$$

$$N_{\mathcal{P}_1} = \#\{(\mathcal{Q}, i, j) : \mathcal{Q} = \mathcal{P}_1, j = 1\}. \quad (7)$$

Then, the probability of correct discrimination between \mathcal{P}_U and \mathcal{P}_1 is given by

$$p_{\text{succ}} = \frac{N_{\mathcal{P}_U} + N_{\mathcal{P}_1}}{N_{\text{total}}}, \quad (8)$$

where N_{total} is the number of trials.

3.2.2. Importance of choosing the optimal discrimination scheme

In principle, the schemes described in the previous section could be used with any choice of $|\psi_0\rangle$ and final measurements \mathcal{P}_{V_i} . However, we argue that it is best to choose those components in such a way that they maximize the probability of correct discrimination. To see that, suppose that some choice of $|\psi_0\rangle, \mathcal{P}_{V_0}, \mathcal{P}_{V_1}$ yields the theoretical upper bound of discriminating between two measurements of one, i.e. on a perfect quantum computer you will always make a correct guess. Then, on real hardware, we might obtain any empirical value in range $[\frac{1}{2}, 1]$. On the other hand, if we choose the components of our scheme such that the successful discrimination probability is only $\frac{3}{5}$, the possible range of empirically obtainable probabilities is only $[\frac{1}{2}, \frac{3}{5}]$. Hence, in the second case, the discrepancy between theoretical and empirical results will be less pronounced.

3.2.3. Constructing optimal discrimination scheme

To construct the optimal discrimination scheme, one starts by calculating the probability of correct discrimination. Using the celebrated result by

171 Helstrom [33], one finds that the optimal probability of correct discrimination
 172 between two quantum measurements, \mathcal{P} and \mathcal{Q} , is

$$p_{\text{succ}}(\mathcal{P}, \mathcal{Q}) = \frac{1}{2} + \frac{1}{4} \|\mathcal{P} - \mathcal{Q}\|_{\diamond}, \quad (9)$$

173 where

$$\|\mathcal{P} - \mathcal{Q}\|_{\diamond} = \max_{\|\psi\|_1=1} \|((\mathcal{P} - \mathcal{Q}) \otimes \mathbb{1})(|\psi\rangle\langle\psi|)\|_1. \quad (10)$$

174 The quantum state $|\psi_0\rangle$ maximizing the diamond norm above is called the
 175 *discriminator*, and can be computed e.g. using semidefinite programming
 176 (SDP) [32, 34]. Furthermore, using the proof of the Holevo-Helstrom theo-
 177 rem, it is possible to construct corresponding unitaries V_0, V_1 to create the
 178 optimal discrimination strategy. For brevity, we do not describe this proce-
 179 dure here. Instead, we refer the interested reader to [32].

180 4. Discrimination scheme for parameterized Fourier family and im- 181 plementation

182 So far, we only discussed how the discrimination is performed assuming
 183 that all needed components $|\psi_0\rangle$, V_0 , and V_1 are known. In this section, we
 184 provide a concrete example using parametrized Fourier family of measure-
 185 ments.

186 The parametrized Fourier family of measurements is defined as a set of
 187 the measurements $\{\mathcal{P}_{U_\phi} : \phi \in [0, 2\pi]\}$, where

$$U_\phi = H \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} H^\dagger, \quad (11)$$

188 and H is the Hadamard matrix of dimension two. For each element of this
 189 set, the discriminator is a Bell state:

$$|\psi_0\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle). \quad (12)$$

190 Observe that $|\psi_0\rangle$ does not depend on the angle ϕ . However, the unitaries
 191 V_0, V_1 depend on ϕ and take the following form:

$$V_0 = \begin{pmatrix} i \sin\left(\frac{\pi-\phi}{4}\right) & -i \cos\left(\frac{\pi-\phi}{4}\right) \\ \cos\left(\frac{\pi-\phi}{4}\right) & \sin\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}, \quad (13)$$

192

$$V_1 = \begin{pmatrix} -i \cos\left(\frac{\pi-\phi}{4}\right) & i \sin\left(\frac{\pi-\phi}{4}\right) \\ \sin\left(\frac{\pi-\phi}{4}\right) & \cos\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}. \quad (14)$$

193 Finally, the theoretical probability of correct discrimination between von
 194 Neumann measurements \mathcal{P}_{U_ϕ} and \mathcal{P}_1 is given by

$$p_{\text{succ}}(\mathcal{P}_{U_\phi}, \mathcal{P}_1) = \frac{1}{2} + \frac{|1 - e^{i\phi}|}{4}. \quad (15)$$

195 We explore the construction of $|\psi_0\rangle$, V_0 and V_1 for parametrized Fourier
 196 family of measurements in Appendix C.

197 5. Software description

198 This section is divided into two parts. In Section 5.1 we describe func-
 199 tionalities of PyQBench package. Next, in Section 5.2, we give a general
 200 overview of the software architecture.

201 5.1. Software Functionalities

202 The PyQBench can be used in two modes: as a Python library and as a
 203 CLI script. When used as a library, PyQBench allows the customization of
 204 discrimination scheme. The user provides a unitary matrix U defining the
 205 measurement to be discriminated, the discriminator $|\psi_0\rangle$, and unitaries V_0
 206 and V_1 describing the final measurement. The PyQBench library provides
 207 then the following functionalities.

- 208 1. Assembling circuits for both postselection and direct sum-based dis-
 209 crimination schemes.
- 210 2. Executing the whole benchmarking scenario on specified backend (ei-
 211 ther real hardware or software simulator).
- 212 3. Interpreting the obtained outputs in terms of discrimination probab-
 213 ilities.

214 Note that the execution of circuits by PyQBench is optional. Instead, the
 215 user might want to opt in for fine-grained control over the execution of the
 216 circuits. For instance, suppose the user wants to simulate the discrimination
 217 experiment on a noisy simulator. In such a case, they can define the necessary
 218 components and assemble the circuits using PyQBench. The circuits can
 219 then be altered, e.g. to add noise to particular gates, and then run using any
 220 Qiskit backend by the user. Finally, PyQBench can be used to interpret the
 221 measurements to obtain discrimination probability.

222 The PyQBench library also contains a readily available implementation
 223 of all necessary components needed to run discrimination experiments for
 224 parametrized Fourier family of measurements, defined previously in Section
 225 4. However, if one only wishes to use this particular family of measurements

in their benchmarks, then using PyQBench as a command line tool might be more straightforward. PyQBench’s command line interface allows running the benchmarking process without writing Python code. The configuration of CLI is done by YAML [35] files describing the benchmark to be performed and the description of the backend on which the benchmark should be run. Notably, the YAML configuration files are reusable. The same benchmark can be used with different backends and vice versa.

The following section describes important architectural decisions taken when creating PyQBench, and how they affect the end-user experience.

5.2. Software Architecture

5.2.1. Overview of the software structure

As already described, PyQBench can be used both as a library and a CLI. Both functionalities are implemented as a part of `qbench` Python package. The exposed CLI tool is also named `qbench`. For brevity, we do not discuss the exact structure of the package here, and instead refer an interested reader to the source code available at GitHub [36] or at the reference manual [37].

PyQBench can be installed from official Python Package Index (PyPI) by running `pip install pyqbench`. In a properly configured Python environment the installation process should also make the `qbench` command available to the user without a need for further configuration.

5.2.2. Integration with hardware providers and software simulators

PyQBench is built around the Qiskit [11] ecosystem. Hence, both the CLI tool and the `qbench` library can use any Qiskit-compatible backend. This includes, IBM Q backends (available by default in Qiskit) and Amazon Braket devices and simulators (available through `qiskit-braket-provider` package [38, 39]).

When using PyQBench as library, instances of Qiskit backends can be passed to functions that expect them as parameters. However, in CLI mode, the user has to provide a YAML file describing the backend. An example of such file can be found in Section 6, and the detailed description of the expected format can be found at PyQBench’s documentation.

5.2.3. Command Line Interface

The Command Line Interface (CLI) of PyQBench has nested structure. The general form of the CLI invocation is shown in listing 1.

Listing 1: Invocation of `qbench` script

```
qbench <benchmark-type> <command> <parameters>
```

263 Currently, PyQBench’s CLI supports only one type of benchmark (discrimi-
264 nation of parametrized Fourier family of measurements), but we decided on
265 structuring the CLI in a hierarchical fashion to allow for future extensions.
266 Thus, the only accepted value of `<benchmark-type>` is `disc-fourier`. The
267 `qbench disc-fourier` command has four subcommands:

- 268 • **benchmark**: run benchmarks. This creates either a result YAML file
269 containing the measurements or an intermediate YAML file for asyn-
270 chronous experiments.
- 271 • **status**: query status of experiments submitted for given benchmark.
272 This command is only valid for asynchronous experiments.
- 273 • **resolve**: query the results of asynchronously submitted experiments
274 and write the result YAML file. The output of this command is almost
275 identical to the result obtained from synchronous experiments.
- 276 • **tabulate**: interpret the results of a benchmark and summarize them
277 in the CSV file.

278 We present usage of each of the above commands later in section 6.

279 5.2.4. *Asynchronous vs. synchronous execution*

280 PyQBench’s CLI can be used in synchronous and asynchronous modes.
281 The mode of execution is defined in the YAML file describing the backend
282 (see Section 6 for an example of this configuration). We decided to couple
283 the mode of execution to the backend description because some backends
284 cannot work in asynchronous mode.

285 When running `qbench disc-fourier benchmark` in asynchronous mode,
286 the PyQBench submits all the circuits needed to perform a benchmark and
287 then writes an intermediate YAML file containing metadata of submitted
288 experiments. In particular, this metadata contains information on correlating
289 submitted job identifiers with particular circuits. The intermediate file can
290 be used to query the status of the submitted jobs or to resolve them, i.e. to
291 wait for their completion and get the measurement outcomes.

292 In synchronous mode, PyQBench first submits all jobs required to run the
293 benchmark and then immediately waits for their completion. The advantage
294 of this approach is that no separate invocation of `qbench` command is needed
295 to actually download the measurement outcomes. The downside, however,
296 is that if the script is interrupted while the command is running, the inter-
297 mediate results will be lost. Therefore, we recommend using asynchronous
298 mode whenever possible.

299 6. Illustrative examples

300 In this section, we present two examples demonstrating the usage of
 301 PyQBench. In the first example, we show how to implement a discrimi-
 302 nation scheme for a user-defined measurement and possible ways of using
 303 this scheme with `qbench` library. The second example demonstrates the us-
 304 age of the CLI. We show how to prepare the input files for the benchmark
 305 and how to run it using the `qbench` tool.

306 6.1. Using user-defined measurement with `qbench` package

307 In this example, we will demonstrate how `qbench` package can be used
 308 with user-defined measurement. For this purpose, we will use $U = H$ (the
 309 Hadamard gate). The detailed calculations that lead to the particular form
 310 of the discriminator and final measurements can be found in Appendix B.
 311 The explicit formula for discriminator in this example reads:

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad (16)$$

312 with final measurements being equal to

$$V_0 = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}, \quad (17)$$

313 and

$$V_1 = \begin{pmatrix} -\beta & \alpha \\ \alpha & \beta \end{pmatrix}, \quad (18)$$

314 where

$$\alpha = \frac{\sqrt{2 - \sqrt{2}}}{2} = \cos\left(\frac{3}{8}\pi\right), \quad (19)$$

315

$$\beta = \frac{\sqrt{2 + \sqrt{2}}}{2} = \sin\left(\frac{3}{8}\pi\right). \quad (20)$$

316 To use the above benchmarking scheme in PyQBench, we first need to con-
 317 struct circuits that can be executed by actual hardware. To this end, we
 318 need to represent each of the unitaries as a sequence of standard gates, keep-
 319 ing in mind that quantum circuits start execution from the $|00\rangle$ state. The
 320 circuit taking $|00\rangle$ to the Bell state $|\psi_0\rangle$ comprises the Hadamard gate fol-
 321 lowed by CNOT gate on both qubits (see Fig. 6). For V_0 and V_1 observe that
 322 $V_0 = \text{RY}\left(\frac{3}{4}\pi\right)$, where RY is rotation gate around the Y axis defined by

$$\text{RY}(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}. \quad (21)$$

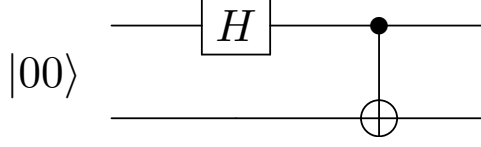


Figure 6: Decomposition of the Bell state $|\psi_0\rangle$.

323 To obtain V_1 we need only to swap the columns, i.e.

$$V_1 = \text{RY} \left(\frac{3}{4}\pi \right) X. \quad (22)$$

324 Finally, the optimal probability of correct discrimination is equal to

$$p_{\text{succ}}(\mathcal{P}_U, \mathcal{P}_1) = \frac{1}{2} + \frac{\sqrt{2}}{4}. \quad (23)$$

325 We will now demonstrate how to implement this theoretical scheme in PyQBench.
 326 For this example we will use the Qiskit Aer simulator [40]. First, we import
 327 the necessary functions and classes from PyQBench and Qiskit. We also im-
 328 port `numpy` for the definition of `np.pi` constant and the exponential function.
 329 The exact purpose of the imported functions will be described at the point
 330 of their usage.

Listing 2: Imports needed for running benchmarking example

```
331 

---


332 import numpy as np
333 from qiskit import QuantumCircuit, Aer
334 from qbench.schemes.postselection import
335     benchmark_using_postselection
336 from qbench.schemes.direct_sum import benchmark_using_direct_sum
337 

---


```

338 To implement the discrimination scheme in PyQBench, we need to define all
 339 the necessary components as Qiskit instructions. We can do so by construct-
 340 ing a circuit object acting on qubits 0 and 1 and then converting them using
 341 `to_instruction()` method.

Listing 3: Defining components for Hadamard experiment

```
342 

---


343 def state_prep():
344     circuit = QuantumCircuit(2)
345     circuit.h(0)
346     circuit.cnot(0, 1)
347     return circuit.to_instruction()
348
349 def u_dag():
```

```

350     circuit = QuantumCircuit(1)
351     circuit.h(0)
352     return circuit.to_instruction()
353
354 def v0_dag():
355     circuit = QuantumCircuit(1)
356     circuit.ry(-np.pi * 3 / 4, 0)
357     return circuit.to_instruction()
358
359 def v1_dag():
360     circuit = QuantumCircuit(1)
361     circuit.ry(-np.pi * 3 / 4, 0)
362     circuit.x(0)
363     return circuit.to_instruction()
364
365 def v0_v1_direct_sum_dag():
366     circuit = QuantumCircuit(2)
367     circuit.ry(-np.pi * 3 / 4, 0)
368     circuit.cnot(0, 1)
369     return circuit.to_instruction()
370

```

371 We now construct a backend object, which in this case is an instance of Aer
372 simulator.

Listing 4: Defining a backend

```

373
374 simulator = Aer.get_backend("aer_simulator")
375

```

376 In the simplest scenario, when one does not want to tweak execution
377 details and simply wishes to run the experiment on a given backend, ev-
378 erything that is required is now to run `benchmark_using_postselection` or
379 `benchmark_using_direct_sum` function, depending on the user preference.

Listing 5: Simulation benchmark by using postselection

```

380
381 postselection_result = benchmark_using_postselection(
382     backend=simulator,
383     target=0,
384     ancilla=1,
385     state_preparation=state_prep(),
386     u_dag=u_dag(),
387     v0_dag=v0_dag(),
388     v1_dag=v1_dag(),
389     num_shots_per_measurement=10000,
390 )

```

391

Listing 6: Simulation benchmark by using direct sum

```

392
393 direct_sum_result = benchmark_using_direct_sum(
394     backend=simulator,
395     target=1,
396     ancilla=2,
397     state_preparation=state_prep(),
398     u_dag=u_dag(),
399     v0_v1_direct_sum_dag=v0_v1_direct_sum_dag(),
400     num_shots_per_measurement=10000,
401 )
402

```

403 The `postselection_result` and `direct_sum_result` variables contain now
404 the empirical probabilities of correct discrimination. We can compare them
405 to the theoretical value and compute the absolute error.

Listing 7: Examining the benchmark results

```

406
407 p_succ = (2 + np.sqrt(2)) / 4
408 print(f"Analytical p_succ = {p_succ}")
409 print(
410     f"Postselection: p_succ = {postselection_result}, abs. error =
411         {p_succ - postselection_result}"
412 )
413 print(f"Direct sum: p_succ = {direct_sum_result}, abs. error =
414     {p_succ - direct_sum_result}")
415

```

```

416
417 Analytical p_succ = 0.8535533905932737
418 Postselection: p_succ = 0.8559797193791593, abs. error =
419     -0.0024263287858855564
420 Direct sum: p_succ = 0.85605, abs. error = -0.0024966094067262468
421

```

422 In the example presented above we used functions that automate the
423 whole process – from the circuit assembly, through running the simulations to
424 interpreting the results. But what if we want more control over some parts of
425 this process? One possibility would be to add some additional parameters to
426 `benchmark_using_xyz` functions, but this approach is not scalable. Moreover,
427 anticipating all possible uses cases is impossible. Therefore, we decided on
428 another approach. PyQBench provides functions performing:

- 429 1. Assembly of circuits needed for experiment, provided the components
430 discussed above.

431 2. Interpretation of the obtained measurements.

432 The difference between the two approaches is illustrated on the diagrams in
433 Fig. 7.

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

Listing 8: Assembling circuits

```
437 

---

438 from qbench.schemes.postselection import (  
439     assemble_postselection_circuits,  
440     compute_probabilities_from_postselection_measurements,  
441 )  
442  
443 circuits = assemble_postselection_circuits(  
444     target=0,  
445     ancilla=1,  
446     state_preparation=state_prep(),  
447     u_dag=u_dag(),  
448     v0_dag=v0_dag(),  
449     v1_dag=v1_dag(),  
450 )  
451 

---


```

452 Recall that for a postselection scheme we have two possible choices of the
453 "unknown" measurement and two possible choices of a final measurement,
454 which gives a total of four circuits needed to run the benchmark. The function
455 `assemble_postselection_circuits` creates all four circuits and places them
456 in a dictionary with keys "id_v0", "id_v1", "u_v0", "u_v1".

457 We will now run our circuits using noisy and noiseless simulation. We
458 start by creating a noise model using Qiskit.

Listing 9: Adding noise model

```
459 

---

460 from qiskit.providers.aer import noise  
461  
462 error = noise.ReadoutError([[0.75, 0.25], [0.8, 0.2]])  
463  
464 noise_model = noise.NoiseModel()  
465 noise_model.add_readout_error(error, [0])  
466 noise_model.add_readout_error(error, [1])  
467 

---


```

468 Once we have our noise model ready, we can execute the circuits with and
469 without noise. To this end, we will use Qiskit's `execute` function. One
470 caveat is that we have to keep track which measurements correspond to

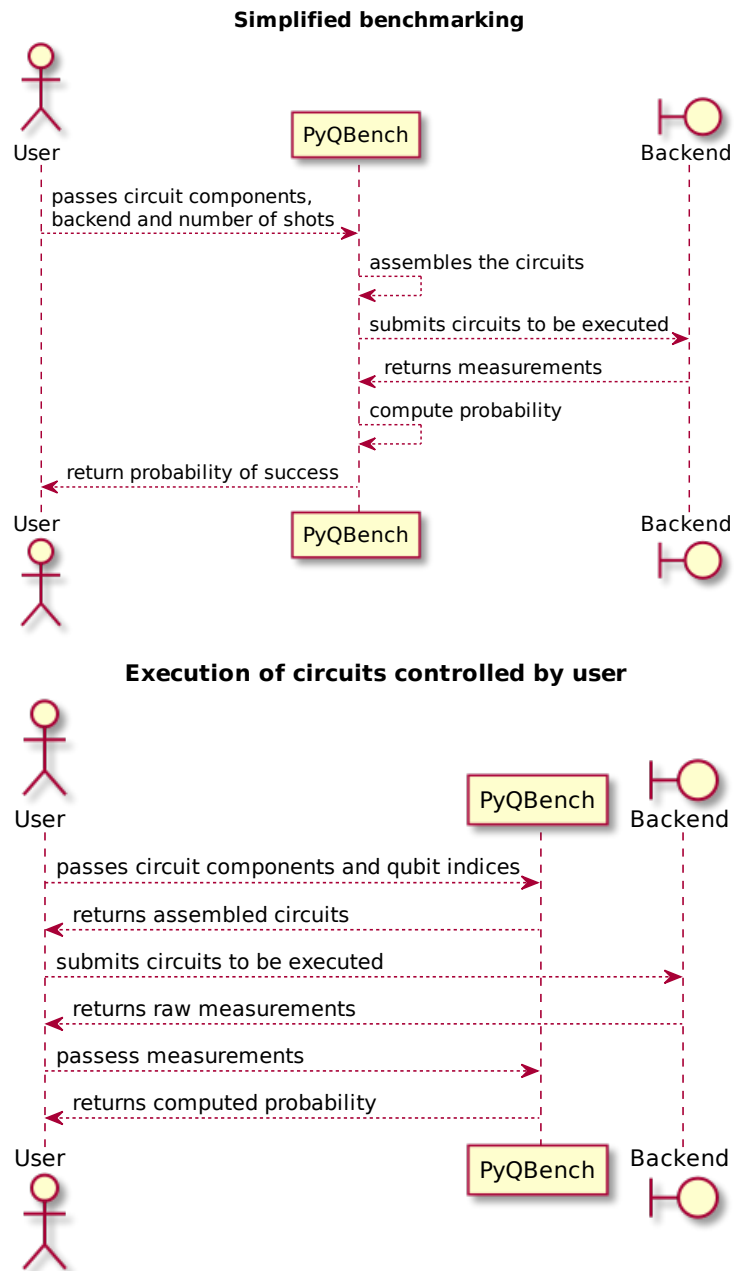


Figure 7: Differences between simplified (top) and user-controlled (bottom) execution of benchmarks in PyQBench. Compared to simplified benchmarking, in user-controlled benchmarks the user has direct access to the circuits being run, and hence can alter them (e.g. by adding noise) and/or choose the parameters used for executing them on the backend.

471 which circuit. We do so by fixing an ordering on the keys in the `circuits`
472 dictionary.

Listing 10: Running circuits

```
473 

---

474 from qiskit import execute
475
476 keys_ordering = ["id_v0", "id_v1", "u_v0", "u_v1"]
477 all_circuits = [circuits[key] for key in keys_ordering]
478
479 counts_noisy = execute(
480     all_circuits,
481     backend=simulator,
482     noise_model=noise_model,
483     shots=10000
484 ).result().get_counts()
485
486 counts_noiseless = execute(
487     all_circuits,
488     backend=simulator,
489     shots=10000
490 ).result().get_counts()
491 

---


```

492 Finally, we use the measurement counts to compute discrimination proba-
493 bilities using `compute_probabilities_from_postselection_measurements`
494 function.

Listing 11: Computation probabilities

```
495 

---

496 prob_succ_noiseless =
497     compute_probabilities_from_postselection_measurements(
498         id_v0_counts=counts_noiseless[0],
499         id_v1_counts=counts_noiseless[1],
500         u_v0_counts=counts_noiseless[2],
501         u_v1_counts=counts_noiseless[3],
502     )
503
504
505 prob_succ_noisy =
506     compute_probabilities_from_postselection_measurements(
507         id_v0_counts=counts_noisy[0],
508         id_v1_counts=counts_noisy[1],
509         u_v0_counts=counts_noisy[2],
510         u_v1_counts=counts_noisy[3],
511     )
```

512

513 We can now examine the results. As an example, in one of our runs, we ob-
 514 tained `prob_succ_noiseless = 0.8524401115559386` and `prob_succ_noisy`
 515 `= 0.5017958400693446`. As expected, for noisy simulations, the result lies
 516 further away from the target value of `0.8535533905932737`.

517 This concludes our example. In the next section, we will show how to use
 518 PyQBench’s CLI.

519 6.2. Using *qbench* CLI

520 Using PyQBench as a library allows one to conduct a two-qubits bench-
 521 mark with arbitrary von Neumann measurement. However, as discussed in
 522 the previous guide, it requires writing some amount of code. For a Fourier
 523 parametrized family of measurements, PyQBench offers a simplified way of
 524 conducting benchmarks using a Command Line Interface (CLI). The work-
 525 flow with PyQBench’s CLI can be summarized as the following list of steps::

- 526 1. Preparing configuration files describing the backend and the experiment
 527 scenario.
- 528 2. Submitting/running experiments. Depending on the experiment sce-
 529 nario, execution can be synchronous, or asynchronous.
- 530 3. (optional) Checking the status of the submitted jobs if the execution
 531 is asynchronous.
- 532 4. Resolving asynchronous jobs into the actual measurement outcomes.
- 533 5. Converting obtained measurement outcomes into tabulated form.

534 6.2.1. Preparing configuration files

535 The configuration of PyQBench CLI is driven by YAML files. The first
 536 configuration file describes the experiment scenario to be executed. The
 537 second file describes the backend. Typically, this backend will correspond to
 538 the physical device to be benchmarked, but for testing purposes one might as
 539 well use any other Qiskit-compatible backend including simulators. Let us
 540 first describe the experiment configuration file, which might look as follow.

Listing 12: Defining the experiment

```

541 type: discrimination-fourier
542
543 qubits:
544   - target: 0
545     ancilla: 1
546   - target: 1
547     ancilla: 2
548 angles:
```

```

549     start: 0
550     stop: 2 * pi
551     num_steps: 3
552 gateset: ibmq
553 method: direct_sum
554 num_shots: 100
555

```

556 The experiment file contains the following fields:

- 557 • **type**: a string describing the type of the experiment. Currently, the
558 only option of **type** is **discrimination-fourier**.

- 559 • **qubits**: a list enumerating pairs of qubits on which the experiment
560 should be run. For configuration in listing 12, the benchmark will run
561 on two pairs of qubits. The first pair is 0 and 1, and the second one is
562 1 and 2. We decided to describe a pair by using **target** and **ancilla**
563 keys rather than using a plain list to emphasize that the role of qubits
564 in the experiment is not symmetric.

- 565 • **angles**: an object describing the range of angles for Fourier param-
566 eterized family. The described range is always uniform, starts at the
567 **start**, ends at **stop** and contains **num_steps** points, including both
568 **start** and **stop**. The **start** and **stop** can be arithmetic expressions
569 using **pi** literal. For instance, the range defined in listing 12 contains
570 three points: 0, π and 2π .

- 571 • **gateset**: a string describing the set of gates used in the decomposi-
572 tion of circuits in the experiment. The PyQBench contains explicit
573 implementations of circuits. The possible options are [**ibmq**, **lucy**,
574 **rigetti**], corresponding to decompositions compatible with IBM Q
575 devices, OQC Lucy device, and Rigetti devices. Alternatively, one
576 might wish to turn off the decomposition by using a special value
577 **generic**. However, for this to work a backend used for the experi-
578 ment must natively implement all the gates needed for the experiment,
579 as described in 4.

- 580 • **method**: a string, either **postselection** or **direct_sum** determining
581 which implementation of the conditional measurement is used.

- 582 • **num_shots**: an integer defines how many shots are performed in the
583 experiment for a particular angle, qubit pair and circuit. Note that if
584 one wishes to compute the total number of shots in the experiment, it
585 is necessary to take into account that the **postselection** method uses
586 twice as many circuits as the **direct_sum** method.

587 The second configuration file describes the backend. We decided to de-
588 couple the experiment and the backend files because it facilitates their reuse.
589 For instance, the same experiment file can be used to run benchmarks on
590 multiple backends, and the same backend description file can be used with
591 multiple experiments.

592 Different Qiskit backends typically require different data for their initial-
593 ization. Hence, there are multiple possible formats of the backend config-
594 uration files understood by PyQBench. We refer the interested reader to
595 the PyQBench’s documentation. Below we describe an example YAML file
596 describing IBM Q backend named Quito.

Listing 13: IBMQ backend

```
597 

---

598 name: ibmq_quito  
599 asynchronous: false  
600 provider:  
601   hub: ibm-q  
602   group: open  
603   project: main  
604 

---


```

605 IBMQ backends typically require an access token to IBM Quantum Experi-
606 ence. Since it would be unsafe to store it in plain text, the token has to be
607 configured separately in `IBMQ_TOKEN` environmental variable.

608 6.2.2. Remarks on using the asynchronous flag

609 For backends supporting asynchronous execution, the `asynchronous` set-
610 ting can be configured to toggle it. For asynchronous execution to work, the
611 following conditions have to be met:

- 612 • Jobs returned by the backend have unique `job_id`.
- 613 • Jobs are retrievable from the backend using the `backend.retrieve_job`
614 method, even from another process (e.g. if the original process running
615 the experiment has finished).

616 Since PyQBench cannot determine if the job retrieval works for a given back-
617 end, it is the user’s responsibility to ensure that this is the case before setting
618 `asynchronous` to `true`.

619 6.2.3. Running the experiment and collecting measurements data

620 After preparing YAML files defining experiment and backend, running the
621 benchmark can be launched by using the following command line invocation:

```
622 

---

623 qbench disc-fourier benchmark experiment_file.yml backend_file.yml  
624 

---


```

625 The output file will be printed to stdout. Optionally, the `--output OUTPUT`
 626 parameter might be provided to write the output to the `OUTPUT` file instead.

```
627
628 qbench disc-fourier benchmark experiment_file.yml backend_file.yml
629     --output async_results.yml
630
```

631 The result of running the above command can be twofold:

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

637 For synchronous experiment, the part of output looks similar to the one
 638 below. The whole YAML file can be seen in Appendix E.

```
639 data:
640 - target: 0
641   ancilla: 1
642   phi: 0.0
643   results_per_circuit:
644   - name: id
645     histogram: {'00': 28, '01': 26, '10': 21, '11': 25}
646     mitigation_info:
647       target: {prob_meas0_prep1: 0.052200000000000024,
648               prob_meas1_prep0: 0.0172}
649       ancilla: {prob_meas0_prep1: 0.059000000000000005,
650                prob_meas1_prep0: 0.0202}
651     mitigated_histogram: {'00': 0.2637212373658018, '01':
652                          0.25865061319892463, '10': 0.2067279352110304, '11':
653                          0.2709002142242433}
654
655
```

656 The data includes `target`, `ancilla`, `phi`, and `results_per_circuit`. The
 657 first three pieces of information have already been described. The last data
 658 `results_per_circuit` gives us the following additional information:

- 659 • **name**: the information which measurement is used during experiment,
 660 either string `"u"` for \mathcal{P}_U or string `"id"` for \mathcal{P}_1 . In this example we
 661 consider \mathcal{P}_1 .
- 662 • **histogram**: the dictionary with measurements' outcomes. The keys
 663 represent possible bitstrings, whereas the values are the number of oc-
 664 currences.

- `mitigation_info`: for some backends (notably for backends corresponding to IBM Q devices), `backends.properties().qubits` contains information that might be used for error mitigation using the MThree method [41, 42]. If this info is available, it will be stored in the `mitigation_info` field, otherwise this field will be absent.
- `mitigated_histogram`: the histogram with measurements' outcomes after the error mitigation.

6.2.4. (Optional) Getting status of asynchronous jobs

PyQBench provides also a helper command that will fetch the statuses of asynchronous jobs. The command is:

```
qbench disc-fourier status async_results.yml
```

and it will display dictionary with histogram of statuses.

6.2.5. Resolving asynchronous jobs

For asynchronous experiments, the stored intermediate data has to be resolved in actual measurements' outcomes. The following command will wait until all jobs are completed and then write a result file.

```
qbench disc-fourier resolve async_results.yml resolved.yml
```

The resolved results, stored in `resolved.yml`, would look just like if the experiment was run synchronously. Therefore, the final results will look the same no matter in which mode the benchmark was run, and hence in both cases the final output file is suitable for being an input for the command computing the discrimination probabilities.

6.2.6. Computing probabilities

As a last step in the processing workflow, the results file has to be passed to `tabulate` command:

```
qbench disc-fourier tabulate results.yml results.csv
```

A sample CSV file is provided below:

7. Impact

With the surge of availability of quantum computing architectures in recent years it becomes increasingly difficult to keep track of their relative performance. To make this case even more difficult, various providers give

target	ancilla	phi	ideal_prob	disc_prob	mit_disc_prob
0	1	0	0.5	0.46	0.45
0	1	3.14	1	0.95	0.98
0	1	6.28	0.5	0.57	0.58
1	2	0	0.5	0.57	0.57
1	2	3.14	1	0.88	0.94
1	2	6.28	0.5	0.55	0.56

Table 2: The resulting CSV file contains table with columns `target`, `ancilla`, `phi`, `ideal_prob`, `disc_prob` and, optionally, `mit_disc_prob`. Each row in the table describes results for a tuple of (`target`, `ancilla`, `phi`). The reference optimal value of discrimination probability is present in `ideal_prob` column, whereas the obtained, empirical discrimination probability can be found in the `disc_prob` column. The `mit_disc_prob` column contains empirical discrimination probability after applying the `Mthree` error mitigation [41, 42], if it was applied.

access to different figures of merit for their architectures. Our package allows the user to test various architectures, available through `qiskit` and Amazon BraKet using problems with simple operational interpretation. We provide one example built-in in the package. Furthermore, we provide a powerful tool for the users to extend the range of available problems in a way that suits their needs.

Due to this possibility of extension, the users are able to test specific aspects of their architecture of interest. For example, if their problem is related to the amount of coherence (the sum of absolute value of off-diagonal elements) of the states present during computation, they are able to quickly prepare a custom experiment, launch it on desired architectures, gather the result, based on which they can decide which specific architecture they should use.

Finally, we provide the source code of PyQBench on GitHub [36] under an open source license which will allow users to utilize and extend our package in their specific applications.

8. Conclusions

In this study, we develop a Python library PyQBench, an innovative open-source framework for benchmarking gate-based quantum computers.

PyQBench can benchmark NISQ devices by verifying their capability of discriminating between two von Neumann measurements. PyQBench offers a simplified, ready-to-use, command line interface (CLI) for running benchmarks using a predefined parameterized Fourier family of measurements. For

725 more advanced scenarios, PyQBench offers a way of employing user-defined
726 measurements instead of predefined ones.

727 9. Conflict of Interest

728 We wish to confirm that there are no known conflicts of interest associated
729 with this publication and there has been no significant financial support for
730 this work that could have influenced its outcome.

731 Acknowledgements

732 This work is supported by the project “Near-term quantum computers
733 Challenges, optimal implementations and applications” under Grant Num-
734 ber POIR.04.04.00-00-17C1/18-00, which is carried out within the Team-Net
735 programme of the Foundation for Polish Science co-financed by the Euro-
736 pean Union under the European Regional Development Fund. PL is also a
737 holder of European Union scholarship through the European Social Fund,
738 grant InterPOWER (POWR.03.05.00-00-Z305).

739 References

- 740 [1] John Preskill. Quantum computing in the nisc era and beyond. *Quan-*
741 *tum*, 2:79, 2018.
- 742 [2] Rigetti computing. <https://www.rigetti.com/>. Accessed on 2023-02-
743 18.
- 744 [3] IBM Quantum. <https://www.ibm.com/quantum>. Accessed on 2023-02-
745 18.
- 746 [4] Oxford quantum. <http://oxfordquantum.org/>. Accessed on 2023-02-
747 18.
- 748 [5] IonQ. <https://ionq.com/>. Accessed on 2023-02-18.
- 749 [6] Xanadu. <https://www.xanadu.ai/>. Accessed on 2023-02-18.
- 750 [7] D-Wave Systems. <https://www.dwavesys.com/>. Accessed on 2023-02-
751 18.
- 752 [8] QuEra. <https://www.quera.com/>. Accessed on 2023-02-18.
- 753 [9] QCS Documentation. <https://docs.rigetti.com/qcs/>. Accessed on
754 2023-02-18.

- [10] PyQuil Documentation. <https://pyquil-docs.rigetti.com/en/stable/>. Accessed on 2023-02-18.
- [11] Qiskit. <https://qiskit.org/>. Accessed on 2023-02-18.
- [12] IBM Quantum Experience. <https://quantum-computing.ibm.com/>. Accessed on 2023-02-18.
- [13] Amazon Braket. <https://aws.amazon.com/braket/>. Accessed on 2023-02-18.
- [14] Zapata Orquestra Platform. <https://www.zapatacomputing.com/orquestra-platform/>. Accessed on 2023-02-18.
- [15] J. Preskill. Quantum computing 40 years later. *arXiv preprint arXiv:2106.10522*, 2021.
- [16] Y. Liu, M. Otten, R. Bassirianjahromi, L. Jiang, and B. Fefferman. Benchmarking near-term quantum computers via random circuit sampling. *arXiv preprint arXiv:2105.05232*, 2021.
- [17] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.
- [18] J. J. Wallman and S. T. Flammia. Randomized benchmarking with confidence. *New Journal of Physics*, 16(10):103032, 2014.
- [19] J. Helsen, I. Roth, E. Onorati, A. H. Werner, and J. Eisert. General framework for randomized benchmarking. *PRX Quantum*, 3(2):020357, 2022.
- [20] A. Cornelissen, J. Bausch, and A. Gilyén. Scalable benchmarks for gate-based quantum computers. *arXiv preprint arXiv:2104.10698*, 2021.
- [21] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3):032328, 2019.
- [22] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

- [23] E. Pelofske, A. Bärtshi, and S. Eidenbenz. Quantum volume in practice: What users can expect from nisq devices. *IEEE Transactions on Quantum Engineering*, 3:1–19, 2022.
- [24] N. Quetschlich, L. Burgholzer, and R. Wille. Mqt bench: Benchmarking software and design automation tools for quantum computing. *arXiv preprint arXiv:2204.13719*, 2022.
- [25] MQTBench. <https://github.com/cda-tum/MQTBench>. Accessed on 2023-02-18.
- [26] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Visslai, X. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong. SupermarQ: A scalable quantum benchmark suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603. IEEE, 2022.
- [27] SupermarQ. <https://github.com/SupertechLabs/SupermarQ>. Accessed on 2023-02-18.
- [28] Qiskit benchmarks. <https://github.com/qiskit-community/qiskit-benchmarks>. Accessed on 2023-02-18.
- [29] Forest Benchmarking: QCVV using PyQuil. <https://github.com/rigetti/forest-benchmarking>. Accessed on 2023-02-18.
- [30] Quantum volume in practice. <https://github.com/lanl/Quantum-Volume-in-Practice>. Accessed on 2023-02-18.
- [31] Z. Puchała, Ł. Paweł, A. Krawiec, and R. Kukulski. Strategies for optimal single-shot discrimination of quantum measurements. *Physical Review A*, 98(4):042103, 2018.
- [32] J. Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018.
- [33] C. W. Helstrom. Quantum detection and estimation theory. *Journal of Statistical Physics*, 1(2):231–252, 1969.
- [34] J. Watrous. Simpler semidefinite programs for completely bounded norms. *Chicago Journal of Theoretical Computer Science*, 8, 2013.
- [35] YAML Ain’t Markup Language (YAML) Version 1.2. <https://yaml.org/spec/1.2.2/>. Accessed on 2023-02-18.

- [36] PyQBench GitHub Repository. <https://github.com/iitis/PyQBench>. Accessed on 2023-02-18.
- [37] PyQBench Documentation. <https://pyqbench.readthedocs.io/en/latest/>. Accessed on 2023-02-18.
- [38] Introducing the Qiskit Provider for Amazon Braket. <https://aws.amazon.com/blogs/quantum-computing/introducing-the-qiskit-provider-for-amazon-braket/>. Accessed on 2023-02-18.
- [39] Qiskit Braket Provider GitHub Repository. <https://github.com/qiskit-community/qiskit-braket-provider>. Accessed on 2023-02-18.
- [40] Qiskit Aer GitHub Repository. <https://github.com/Qiskit/qiskit-aer>. Accessed on 2023-02-18.
- [41] mthree Documentation. <https://qiskit.org/documentation/partners/mthree/stubs/mthree.M3Mitigation.html>. Accessed on 2023-02-10.
- [42] P. D. Nation, H. Kang, N. Sundaresan, and J. M. Gambetta. Scalable mitigation of measurement errors on quantum computers. *PRX Quantum*, 2(4):040326, 2021.
- [43] F. D. Murnaghan. On the field of values of a square matrix. *Proceedings of the National Academy of Sciences*, 18(3):246–248, 1932.
- [44] Numerical shadow. <https://numericalshadow.org/>. Accessed on 2022-10-02.
- [45] P. Lewandowska, A. Krawiec, R. Kukulski, L. Pawela, and Z. Puchała. On the optimal certification of von Neumann measurements. *Scientific Reports*, 11(1):3623, 2021.
- [46] F. Hausdorff. Der wertvorrat einer bilinearform. *Mathematische Zeitschrift*, 3(1):314–316, 1919.
- [47] O. Toeplitz. Das algebraische analogon zu einem satze von fejér. *Mathematische Zeitschrift*, 2(1-2):187–197, 1918.

850 Appendix A. Mathematical preliminaries

851 Let \mathcal{M}_{d_1, d_2} be the set of all matrices of dimension $d_1 \times d_2$ over the field \mathbb{C} .
 852 For simplicity, square matrices will be denoted by \mathcal{M}_d . By Ω_d , we will denote
 853 the set of quantum states, that is positive semidefinite operators having trace
 854 equal to one. The subset of \mathcal{M}_d consisting of unitary matrices will be denoted
 855 by \mathcal{U}_d , while its subgroup of diagonal unitary operators will be denoted by
 856 \mathcal{DU}_d .

857 We will also need a linear mapping transforming \mathcal{M}_{d_1} into \mathcal{M}_{d_2} , which
 858 will be denoted

$$\Phi : \mathcal{M}_{d_1} \rightarrow \mathcal{M}_{d_2}. \quad (\text{A.1})$$

859 There exists a bijection between the set of linear mappings Φ and the set of
 860 matrices $\mathcal{M}_{d_1 d_2}$, known as the Choi-Jamiołkowski isomorphism. For a given
 861 linear mapping Φ the corresponding Choi operator $J(\Phi)$ is explicitly written
 862 as

$$J(\Phi) := \sum_{i,j=0}^{d-1} \Phi(|i\rangle\langle j|) \otimes |i\rangle\langle j|. \quad (\text{A.2})$$

863 We also introduce a special subset of all mappings Φ , called quantum
 864 channels, which are completely positive and trace preserving (CPTP). In
 865 this work we will consider a special class of quantum channels, called unitary
 866 channels. A quantum channel Φ_U is said to be a unitary channel if it has the
 867 following form $\Phi_U(\cdot) = U \cdot U^\dagger$ for any $U \in \mathcal{U}_d$.

868 Let us recall a general form of a quantum measurement, so called Positive
 869 Operator Valued Measure (POVM). A POVM \mathcal{P} is a collection of positive
 870 semidefinite operators $\{E_1, \dots, E_m\}$, called effects, that sum up to the iden-
 871 tity operator, *i.e.* $\sum_{i=1}^m E_i = \mathbb{1}$. In PyQBench, we are interested only in von
 872 Neumann measurements, that is measurements for which all the effects are
 873 rank-one projectors. Every such measurement can be parameterized by a
 874 unitary matrix $U \in \mathcal{U}_d$ with effects $\{|u_0\rangle\langle u_0|, \dots, |u_{d-1}\rangle\langle u_{d-1}|\}$, are created
 875 by taking $|u_i\rangle$ as $(i+1)$ -th column of the unitary matrix U . We will denote
 876 von Neumann measurements described by the matrix U by \mathcal{P}_U . The action
 877 of von Neumann measurement \mathcal{P}_U on some state $\rho \in \Omega_d$ can be seen as a
 878 measure-and-prepare quantum channel as follows

$$\mathcal{P}_U : \rho \rightarrow \sum_{i=0}^{d-1} \langle u_i | \rho | u_i \rangle |i\rangle\langle i|. \quad (\text{A.3})$$

879 Moreover, observe that each von Neumann measurement \mathcal{P}_U poses a com-
 880 position of a unitary channel Φ_{U^\dagger} and the maximally dephasing channel Δ ,
 881 that means $\mathcal{P}_U = \Delta \circ \Phi_{U^\dagger}$.

882 We need to also briefly discuss about the distance between quantum op-
 883 erations. From [31, Theorem 1], the distance between measurements \mathcal{P}_U and
 884 \mathcal{P}_1 can be expressed in the notion of diamond norm, that is

$$\|\mathcal{P}_U - \mathcal{P}_1\|_\diamond = \min_{E \in \mathcal{DU}_d} \|\Phi_{UE} - \Phi_1\|_\diamond. \quad (\text{A.4})$$

885 To express the distance between unitary channels, we need to introduce the
 886 definition of numerical range [43]. The set

$$W(A) = \{\langle x|A|x\rangle : |x\rangle \in \mathbb{C}^d, \langle x|x\rangle = 1\} \quad (\text{A.5})$$

887 is called the numerical range of a given matrix $A \in \mathcal{M}_d$. The detailed
 888 properties of the numerical range and its generalizations we can read on the
 889 website [44].

890 Due to the definition of $W(A)$, the distance between two unitary channels
 891 Φ_U and Φ_1 can be written as

$$\|\Phi_U - \Phi_1\|_\diamond = 2\sqrt{1 - \nu^2}, \quad (\text{A.6})$$

892 where $\nu = \min_{x \in W(U^\dagger)} |x|$.

893 Appendix B. Discrimination task for Hadamard gate

894 For the discrimination task between von Neumann measurements \mathcal{P}_U and
 895 \mathcal{P}_1 , where $U = H$ (the Hadamard gate), the key is to calculate the diamond
 896 norm $\|\mathcal{P}_H - \mathcal{P}_1\|_\diamond$ and determine the discriminator $|\psi_0\rangle$. Using semidefinite
 897 programming [34], we obtain

$$\|\mathcal{P}_H - \mathcal{P}_1\|_\diamond = \sqrt{2}. \quad (\text{B.1})$$

898 From [45] we have

$$\|\mathcal{P}_H - \mathcal{P}_1\|_\diamond = \|\Phi_{HE_0} - \Phi_1\|_\diamond, \quad (\text{B.2})$$

899 where Φ_U is a unitary channel and E_0 is of the form

$$E_0 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1+i & 0 \\ 0 & -1-i \end{pmatrix}. \quad (\text{B.3})$$

900 Next, in order to construct the discriminator $|\psi_0\rangle$ we use Lemma 5 and the
 901 proof of Theorem 1 in [31]. We show that there exist states ρ_1 and ρ_2 of the
 902 form $\rho_1 = \frac{1}{2} \begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix}$ and $\rho_2 = \frac{1}{2} \begin{pmatrix} 1 & -i \\ i & 1 \end{pmatrix}$, respectively. Thus, we construct
 903 the quantum state ρ_0 as follows:

$$\rho_0 = \frac{1}{2}\rho_1 + \frac{1}{2}\rho_2 = \frac{1}{2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \quad (\text{B.4})$$

904 According to the Lemma 5 and the proof of Theorem 1 in [31] we assume
 905 that

$$|\psi_0\rangle = \left| \sqrt{\rho_0^\top} \right\rangle. \quad (\text{B.5})$$

906 It directly implies that

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \quad (\text{B.6})$$

907 Next, from Holevo-Helstrom theorem [32], we determine the final measure-
 908 ment \mathcal{P}_{V_i} . Let us consider

$$X = (\mathcal{P}_H \otimes \mathbb{1})(|\psi_0\rangle\langle\psi_0|) - (\mathcal{P}_1 \otimes \mathbb{1})(|\psi_0\rangle\langle\psi_0|). \quad (\text{B.7})$$

909 From the Hahn-Jordan decomposition [32], let us note

$$X = P - Q, \quad (\text{B.8})$$

910 where $P, Q \geq 0$. Let us define projectors Π_P and Π_Q onto $\text{im}(P)$ and $\text{im}(Q)$,
 911 respectively. Observe, that P and Q are block-diagonal. Then, Π_P and Π_Q
 912 have the following forms

$$\Pi_P = \begin{pmatrix} |x_p\rangle\langle x_p| & 0 \\ 0 & |y_p\rangle\langle y_p| \end{pmatrix}, \quad (\text{B.9})$$

913 and

$$\Pi_Q = \begin{pmatrix} |x_q\rangle\langle x_q| & 0 \\ 0 & |y_q\rangle\langle y_q| \end{pmatrix}. \quad (\text{B.10})$$

914 Hence, we define V_0 as

$$\begin{cases} |x_p\rangle = V_0|0\rangle \\ |x_q\rangle = V_0|1\rangle \end{cases} \quad (\text{B.11})$$

915 and V_1 as

$$\begin{cases} |y_p\rangle = V_1|0\rangle \\ |y_q\rangle = V_1|1\rangle \end{cases}. \quad (\text{B.12})$$

916 For the discrimination task between \mathcal{P}_H and \mathcal{P}_1 the explicit form of V_0 and
 917 V_1 is given as follows (see also `mathematics/optimal_final_measurement_`
 918 `discrimination.nb` in the source code repository):

$$V_0 = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}, \quad (\text{B.13})$$

919 and

$$V_1 = \begin{pmatrix} -\beta & \alpha \\ \alpha & \beta \end{pmatrix}, \quad (\text{B.14})$$

920 where

$$\alpha = \frac{\sqrt{2 - \sqrt{2}}}{2} = \cos\left(\frac{3}{8}\pi\right), \quad (\text{B.15})$$

921 and

$$\beta = \frac{\sqrt{2 + \sqrt{2}}}{2} = \sin\left(\frac{3}{8}\pi\right). \quad (\text{B.16})$$

922 Appendix C. Optimal probability for parameterized Fourier fam- 923 ily

924 Let us focus on single-qubit von Neumann measurements \mathcal{P}_1 and \mathcal{P}_U .
925 Assume that the unitary matrix U is of the form

$$U = H \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} H^\dagger \quad (\text{C.1})$$

926 where H is the Hadamard matrix of dimension two and $\phi \in [0, 2\pi)$. In this
927 section we present theoretical probability of correct discrimination between
928 these measurements. To do that, we will present an auxiliary lemma.

929 **Lemma 1.** *Let $U = H \text{diag}(1, e^{i\phi}) H^\dagger$, $\phi \in [0, 2\pi)$ and let Φ_U and Φ_1 be two*
930 *unitary channels. Then, the following equation holds*

$$\min_{E \in \mathcal{DU}_2} \|\Phi_{UE} - \Phi_1\|_\diamond = \|\Phi_U - \Phi_1\|_\diamond, \quad (\text{C.2})$$

931 *Proof.* Recall that the distance between two unitary channels is given by
932 $\|\Phi_U - \Phi_1\|_\diamond = 2\sqrt{1 - \nu^2}$, where $\nu = \min_{x \in W(U^\dagger)} |x|$ for any $U \in \mathcal{U}_d$. For
933 $U = H \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} H^\dagger$ the readers briefly observe that $\nu^2 = 1 - \frac{|1 - e^{-i\phi}|^2}{4} =$
934 $1 - \frac{|1 - e^{i\phi}|^2}{4}$. So,

$$\|\Phi_U - \Phi_1\|_\diamond = |1 - e^{i\phi}|. \quad (\text{C.3})$$

935 It implies that it is enough to prove

$$\min_{E \in \mathcal{DU}_2} \|\Phi_{UE} - \Phi_1\|_\diamond = |1 - e^{i\phi}|. \quad (\text{C.4})$$

936 This condition is equivalent to show that for every $E \in \mathcal{DU}_2$

$$\nu_E \leq \frac{|1 + e^{i\phi}|}{2}, \quad (\text{C.5})$$

937 where $\nu_E = \min_{x \in W(U^\dagger E)} |x|$.

938 The celebrated Hausdorff-Töplitz theorem [46, 47] states that $W(A)$ of
939 any matrix $A \in \mathcal{M}_d$ is a convex set, and therefore we have

$$W(A) = \{\text{tr}(A\rho) : \rho \in \Omega_d\}. \quad (\text{C.6})$$

940 So, we can assume that

$$\min_{|x\rangle \in \mathbb{C}^2: |x\rangle\langle x|=1} |\langle x|U^\dagger|x\rangle| = \min_{\rho \in \Omega_2} |\text{tr}(U^\dagger\rho)|. \quad (\text{C.7})$$

941 Then, we have

$$\nu_E = \min_{\rho \in \Omega_2} |\text{tr}(\rho U E)|. \quad (\text{C.8})$$

942 For that, our task is reduced to show that for every $E \in \mathcal{DU}_2$ there exists
943 $\rho \in \Omega_2$ such that

$$|\text{tr}(\rho U E)| \leq \frac{|1 + e^{i\phi}|}{2}. \quad (\text{C.9})$$

944 Let us define $E = \begin{pmatrix} E_0 & 0 \\ 0 & E_1 \end{pmatrix}$ and take $\rho = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$. From spectral
945 theorem, let us decompose U as

$$U = \lambda_0 |x_0\rangle\langle x_0| + \lambda_1 |x_1\rangle\langle x_1|, \quad (\text{C.10})$$

946 where for eigenvalue $\lambda_0 = 1$, the corresponding eigenvector is of the form
947 $|x_0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$, whereas for $\lambda_1 = e^{i\phi}$ we have $|x_1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$. Then, for
948 every $E \in \mathcal{DU}_2$ we have

$$\begin{aligned} |\text{tr}(\rho U E)| &= \frac{1}{2} |\text{tr}(H \text{diag}(1, e^{i\phi}) H^\dagger E)| = \frac{1}{2} |\text{tr}((|x_0\rangle\langle x_0| + e^{i\phi} |x_1\rangle\langle x_1|) E)| \\ &= \frac{1}{2} |\langle x_0|E|x_0\rangle + e^{i\phi} \langle x_1|E|x_1\rangle| = \frac{1}{2} \left| \frac{E_0 + E_1}{2} + e^{i\phi} \frac{E_0 + E_1}{2} \right| \\ &= \frac{|1 + e^{i\phi}|}{2} \left| \frac{E_0 + E_1}{2} \right| \leq \frac{|1 + e^{i\phi}|}{2}, \end{aligned} \quad (\text{C.11})$$

949 which completes the proof. \square

950 **Theorem 1.** *The optimal probability of correct discrimination between von*
951 *Neumann measurements \mathcal{P}_U and \mathcal{P}_1 for $U = H \text{diag}(1, e^{i\phi}) H^\dagger$, where $\phi \in$*
952 *$[0, 2\pi)$ is given by*

$$p_{\text{succ}}(\mathcal{P}_U, \mathcal{P}_1) = \frac{1}{2} + \frac{|1 - e^{i\phi}|}{4}. \quad (\text{C.12})$$

953 *Proof.* From Holevo-Helstrom theorem, we obtain

$$p_{\text{succ}}(\mathcal{P}_U, \mathcal{P}_1) = \frac{1}{2} + \frac{1}{4} \|\mathcal{P}_U - \mathcal{P}_1\|_{\diamond}. \quad (\text{C.13})$$

954 From [31, Theorem 1], we have

$$\|\mathcal{P}_U - \mathcal{P}_1\|_{\diamond} = \min_{E \in \mathcal{DU}_d} \|\Phi_{UE} - \Phi_1\|_{\diamond}. \quad (\text{C.14})$$

955 From Lemma 1, we know that for $U = H \text{diag}(1, e^{i\phi}) H^{\dagger}$, it also holds that

$$\min_{E \in \mathcal{DU}_2} \|\Phi_{UE} - \Phi_1\|_{\diamond} = \|\Phi_U - \Phi_1\|_{\diamond}, \quad (\text{C.15})$$

956 which is exactly equal to

$$\|\Phi_U - \Phi_1\|_{\diamond} = 2\sqrt{1 - \nu^2} = |1 - e^{i\phi}|. \quad (\text{C.16})$$

957 It implies that

$$p_{\text{succ}}(\mathcal{P}_U, \mathcal{P}_1) = \frac{1}{2} + \frac{|1 - e^{i\phi}|}{4}, \quad (\text{C.17})$$

958 which completes the proof. \square

959 **Appendix D. Optimal discrimination strategy for parameterized** 960 **Fourier family**

961 In this Appendix we create the optimal theoretical strategy of discrimi-
962 nation between \mathcal{P}_U and \mathcal{P}_1 . To indicate the optimal strategy, we will present
963 two propositions. The first one is concentrated around the discriminator as
964 the optimal input state of discrimination strategy, whereas the second one
965 describes the optimal final measurement.

966 **Proposition 1.** *Consider the problem of discrimination between von Neu-*
967 *mann measurements \mathcal{P}_U and \mathcal{P}_1 , $U = H \text{diag}(1, e^{i\phi}) H^{\dagger}$ and $\phi \in [0, 2\pi)$. The*
968 *discriminator has the form*

$$|\psi_0\rangle = \frac{1}{\sqrt{2}} |\mathbb{1}_2\rangle. \quad (\text{D.1})$$

969 *Proof.* Let $U = H \text{diag}(1, e^{i\phi}) H^{\dagger}$, $\phi \in [0, 2\pi)$ be decomposed as

$$U = \lambda_0 |x_0\rangle\langle x_0| + \lambda_1 |x_1\rangle\langle x_1|, \quad (\text{D.2})$$

970 where for eigenvalue $\lambda_0 = 1$, the corresponding eigenvector is of the form
 971 $|x_0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$, whereas for $\lambda_1 = e^{i\phi}$ we have $|x_1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$. For Hermitian-
 972 preserving maps [32] the diamond norm may be expressed as

$$\|\Phi\|_\diamond = \max_{\rho \in \Omega_d} \|(\mathbb{1} \otimes \sqrt{\rho}) J(\Phi) (\mathbb{1} \otimes \sqrt{\rho})\|_1. \quad (\text{D.3})$$

973 Hence, we obtain

$$\begin{aligned} \|\mathcal{P}_U - \mathcal{P}_1\|_\diamond &= \max_{\rho \in \Omega_2} \|(\mathbb{1} \otimes \sqrt{\rho}) J(\mathcal{P}_U - \mathcal{P}_1) (\mathbb{1} \otimes \sqrt{\rho})\|_1 \\ &= \max_{\rho \in \Omega_2} \left\| (\mathbb{1} \otimes \sqrt{\rho}) \sum_{i=0}^1 |i\rangle\langle i| \otimes (|u_i\rangle\langle u_i| - |i\rangle\langle i|)^\top (\mathbb{1} \otimes \sqrt{\rho}) \right\|_1 \\ &= \max_{\rho \in \Omega_2} \left\| \sum_{i=0}^1 |i\rangle\langle i| \otimes \sqrt{\rho} (|u_i\rangle\langle u_i| - |i\rangle\langle i|)^\top \sqrt{\rho} \right\|_1 \\ &= \max_{\rho \in \Omega_2} \sum_{i=0}^1 \left\| \sqrt{\rho} (|u_i\rangle\langle u_i| - |i\rangle\langle i|)^\top \sqrt{\rho} \right\|_1. \end{aligned} \quad (\text{D.4})$$

974 One can prove that for all $\alpha, \beta \geq 0$, and unit vectors $|x\rangle, |y\rangle$ the following
 975 equation holds [32]

$$\|\alpha|x\rangle\langle x| - \beta|y\rangle\langle y|\|_1 = \sqrt{(\alpha + \beta)^2 - 4\alpha\beta|\langle x|y\rangle|^2}. \quad (\text{D.5})$$

976 By taking $|x\rangle = \frac{\sqrt{\rho}|\bar{u}_i\rangle}{\|\sqrt{\rho}|\bar{u}_i\rangle\|}$ and $|y\rangle = \frac{\sqrt{\rho}|i\rangle}{\|\sqrt{\rho}|i\rangle\|}$ we have

$$\|\mathcal{P}_U - \mathcal{P}_1\|_\diamond = \max_{\rho \in \Omega_2} \sum_{i=0}^1 \sqrt{(\langle \bar{u}_i|\rho|\bar{u}_i\rangle + \langle i|\rho|i\rangle)^2 - 4|\langle \bar{u}_i|\rho|i\rangle|^2}. \quad (\text{D.6})$$

977 Let us take $\rho_0 = \frac{1}{2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, we obtain

$$\begin{aligned}
\|\mathcal{P}_U - \mathcal{P}_1\|_\diamond &\geq \sum_{i=0}^1 \sqrt{(\langle \bar{u}_i | \rho_0 | \bar{u}_i \rangle + \langle i | \rho_0 | i \rangle)^2 - 4|\langle i | \rho_0 | \bar{u}_i \rangle|^2} \\
&= \sum_{i=0}^1 \sqrt{1 - |\langle i | U | i \rangle|^2} \\
&= \sum_{i=0}^1 \sqrt{1 - |1 \cdot \langle i | u_0 \rangle \langle u_0 | i \rangle + e^{i\phi} \cdot \langle i | u_1 \rangle \langle u_1 | i \rangle|^2} \quad (\text{D.7}) \\
&= \sum_{i=0}^1 \sqrt{1 - \left| \frac{1 + e^{i\phi}}{2} \right|^2} = 2\sqrt{1 - \left| \frac{1 + e^{i\phi}}{2} \right|^2} \\
&= |1 - e^{i\phi}|.
\end{aligned}$$

978 Due to Theorem 1 and the following equality

$$\|(\mathbb{1} \otimes \sqrt{\rho}) J(\mathcal{P}_U - \mathcal{P}_1) (\mathbb{1} \otimes \sqrt{\rho})\|_1 = \left\| ((\mathcal{P}_U - \mathcal{P}_1) \otimes \mathbb{1}) \left(|\sqrt{\rho}^\top\rangle\rangle \langle\langle \sqrt{\rho}^\top| \right) \right\|_1, \quad (\text{D.8})$$

979 the discriminator $|\psi_0\rangle$ is equal to

$$|\psi_0\rangle = |\sqrt{\rho_0}^\top\rangle\rangle = \frac{1}{\sqrt{2}}|\mathbb{1}_2\rangle\rangle, \quad (\text{D.9})$$

980 which completes the proof. \square

981 **Proposition 2.** Consider the problem of discrimination between von Neu-
982 mann measurements \mathcal{P}_U and \mathcal{P}_1 , $U = H \text{diag}(1, e^{i\phi}) H^\dagger$ and $\phi \in [0, 2\pi)$. The
983 controlled unitaries V_0 and V_1 have the form

$$V_0 = \begin{pmatrix} i \sin\left(\frac{\pi-\phi}{4}\right) & -i \cos\left(\frac{\pi-\phi}{4}\right) \\ \cos\left(\frac{\pi-\phi}{4}\right) & \sin\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}, \quad (\text{D.10})$$

984 and

$$V_1 = \begin{pmatrix} -i \cos\left(\frac{\pi-\phi}{4}\right) & i \sin\left(\frac{\pi-\phi}{4}\right) \\ \sin\left(\frac{\pi-\phi}{4}\right) & \cos\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}. \quad (\text{D.11})$$

985 *Proof.* From Proposition 1 we obtain the exact form of discriminator given
986 by

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}|\mathbb{1}_2\rangle\rangle. \quad (\text{D.12})$$

Repeating the procedure used to distinguish the von Neumann measurements in the Hadamard basis (see Appendix B), we use the Hahn-Jordan decomposition and then we create the projective operators into the positive and negative part of X matrix. Hence, the explicit form of V_0 and V_1 is given as follows:

$$V_0 = \begin{pmatrix} i \sin\left(\frac{\pi-\phi}{4}\right) & -i \cos\left(\frac{\pi-\phi}{4}\right) \\ \cos\left(\frac{\pi-\phi}{4}\right) & \sin\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}, \quad (\text{D.13})$$

and

$$V_1 = \begin{pmatrix} -i \cos\left(\frac{\pi-\phi}{4}\right) & i \sin\left(\frac{\pi-\phi}{4}\right) \\ \sin\left(\frac{\pi-\phi}{4}\right) & \cos\left(\frac{\pi-\phi}{4}\right) \end{pmatrix}, \quad (\text{D.14})$$

where $\phi \in [0, 2\pi)$. \square

Appendix E. Output YAML files

In this appendix we present examples of YAML's files obtained from synchronous and asynchronous experiments. We will start at synchronous case.

Listing 14: Defining experiment file

```

type: discrimination-fourier
qubits:
  - target: 0
    ancilla: 1
  - target: 1
    ancilla: 2
angles:
  start: 0
  stop: 2 * pi
  num_steps: 3
gateset: ibmq
method: direct_sum
num_shots: 100

```

Listing 15: Defining backend file

```

name: ibmq_quito
asynchronous: false
provider:
  hub: ibm-q
  group: open
  project: main

```

Listing 16: Results (synchronous)

```

1021
1022 metadata:
1023     experiments:
1024         type: discrimination-fourier
1025         qubits:
1026             - {target: 0, ancilla: 1}
1027             - {target: 1, ancilla: 2}
1028         angles: {start: 0.0, stop: 6.283185307179586, num_steps: 3}
1029         gateset: ibmq
1030         method: direct_sum
1031         num_shots: 100
1032     backend_description:
1033         name: ibmq_quito
1034         asynchronous: false
1035         provider: {group: open, hub: ibm-q, project: main}
1036 data:
1037 - target: 0
1038   ancilla: 1
1039   phi: 0.0
1040   results_per_circuit:
1041   - name: id
1042     histogram: {'00': 28, '01': 26, '10': 21, '11': 25}
1043     mitigation_info:
1044         target: {prob_meas0_prep1: 0.052200000000000024,
1045                 prob_meas1_prep0: 0.0172}
1046         ancilla: {prob_meas0_prep1: 0.059000000000000005,
1047                  prob_meas1_prep0: 0.0202}
1048         mitigated_histogram: {'00': 0.2637212373658018, '01':
1049                               0.25865061319892463, '10': 0.2067279352110304, '11':
1050                               0.2709002142242433}
1051   - name: u
1052     histogram: {'00': 30, '01': 16, '10': 28, '11': 26}
1053     mitigation_info:
1054         target: {prob_meas0_prep1: 0.052200000000000024,
1055                 prob_meas1_prep0: 0.0172}
1056         ancilla: {prob_meas0_prep1: 0.059000000000000005,
1057                  prob_meas1_prep0: 0.0202}
1058         mitigated_histogram: {'00': 0.2857378991684036, '01':
1059                               0.14975297832942433, '10': 0.28142307224788693, '11':
1060                               0.2830860502542851}
1061 - target: 0
1062   ancilla: 1
1063   phi: 3.141592653589793

```

```

1064 results_per_circuit:
1065 - name: id
1066   histogram: {'00': 4, '01': 5, '10': 45, '11': 46}
1067   mitigation_info:
1068     target: {prob_meas0_prep1: 0.052200000000000024,
1069             prob_meas1_prep0: 0.0172}
1070     ancilla: {prob_meas0_prep1: 0.059000000000000005,
1071              prob_meas1_prep0: 0.0202}
1072     mitigated_histogram: {'00': 0.011053610583159325, '01':
1073                          0.02261276648026373, '10': 0.4593955619936729, '11':
1074                          0.5069380609429042}
1075 - name: u
1076   histogram: {'00': 56, '01': 43, '10': 1}
1077   mitigation_info:
1078     target: {prob_meas0_prep1: 0.052200000000000024,
1079             prob_meas1_prep0: 0.0172}
1080     ancilla: {prob_meas0_prep1: 0.059000000000000005,
1081              prob_meas1_prep0: 0.0202}
1082     mitigated_histogram: {'00': 0.5573987337172156, '01':
1083                          0.44424718645642625, '10': -0.0016459201736417181}
1084 - target: 0
1085   ancilla: 1
1086   phi: 6.283185307179586
1087 results_per_circuit:
1088 - name: id
1089   histogram: {'00': 36, '01': 18, '10': 25, '11': 21}
1090   mitigation_info:
1091     target: {prob_meas0_prep1: 0.052200000000000024,
1092             prob_meas1_prep0: 0.0172}
1093     ancilla: {prob_meas0_prep1: 0.059000000000000005,
1094              prob_meas1_prep0: 0.0202}
1095     mitigated_histogram: {'00': 0.3488190312089973, '01':
1096                          0.17355281935572894, '10': 0.2505792064871127, '11':
1097                          0.22704894294816103}
1098 - name: u
1099   histogram: {'00': 32, '01': 27, '10': 24, '11': 17}
1100   mitigation_info:
1101     target: {prob_meas0_prep1: 0.052200000000000024,
1102             prob_meas1_prep0: 0.0172}
1103     ancilla: {prob_meas0_prep1: 0.059000000000000005,
1104              prob_meas1_prep0: 0.0202}
1105     mitigated_histogram: {'00': 0.3025357275361897, '01':
1106                          0.27413673119534815, '10': 0.24313373302688793, '11':

```



```

1107         0.18019380824157433}
1108 - target: 1
1109 ancilla: 2
1110 phi: 0.0
1111 results_per_circuit:
1112 - name: id
1113   histogram: {'00': 27, '01': 20, '10': 24, '11': 29}
1114   mitigation_info:
1115     target: {prob_meas0_prep1: 0.059000000000000005,
1116             prob_meas1_prep0: 0.0202}
1117     ancilla: {prob_meas0_prep1: 0.075400000000000002,
1118              prob_meas1_prep0: 0.0528}
1119     mitigated_histogram: {'00': 0.2594378169217188, '01':
1120                          0.19318893233269735, '10': 0.23035366874292057, '11':
1121                          0.3170195820026633}
1122 - name: u
1123   histogram: {'00': 31, '01': 24, '10': 23, '11': 22}
1124   mitigation_info:
1125     target: {prob_meas0_prep1: 0.059000000000000005,
1126             prob_meas1_prep0: 0.0202}
1127     ancilla: {prob_meas0_prep1: 0.075400000000000002,
1128              prob_meas1_prep0: 0.0528}
1129     mitigated_histogram: {'00': 0.30056875246775644, '01':
1130                          0.2438221628798003, '10': 0.22180309809696985, '11':
1131                          0.23380598655547338}
1132 - target: 1
1133 ancilla: 2
1134 phi: 3.141592653589793
1135 results_per_circuit:
1136 - name: id
1137   histogram: {'00': 5, '01': 4, '10': 50, '11': 41}
1138   mitigation_info:
1139     target: {prob_meas0_prep1: 0.059000000000000005,
1140             prob_meas1_prep0: 0.0202}
1141     ancilla: {prob_meas0_prep1: 0.075400000000000002,
1142              prob_meas1_prep0: 0.0528}
1143     mitigated_histogram: {'00': 0.009552870928837118, '01':
1144                          0.007194089383161034, '10': 0.5236791012692514, '11':
1145                          0.4595739384187503}
1146 - name: u
1147   histogram: {'00': 41, '01': 51, '10': 3, '11': 5}
1148   mitigation_info:
1149     target: {prob_meas0_prep1: 0.059000000000000005,

```

```

1150         prob_meas1_prep0: 0.0202}
1151     ancilla: {prob_meas0_prep1: 0.075400000000000002,
1152               prob_meas1_prep0: 0.0528}
1153     mitigated_histogram: {'00': 0.4073387714165384, '01':
1154                           0.5614614121117936, '10': 0.006431862814564833, '11':
1155                           0.024767953657102992}
1156 - target: 1
1157   ancilla: 2
1158   phi: 6.283185307179586
1159   results_per_circuit:
1160   - name: id
1161     histogram: {'00': 30, '01': 28, '10': 23, '11': 19}
1162     mitigation_info:
1163       target: {prob_meas0_prep1: 0.059000000000000005,
1164               prob_meas1_prep0: 0.0202}
1165       ancilla: {prob_meas0_prep1: 0.075400000000000002,
1166               prob_meas1_prep0: 0.0528}
1167       mitigated_histogram: {'00': 0.2868459834940102, '01':
1168                           0.2919564941384742, '10': 0.22466574543735374, '11':
1169                           0.19653177693016174}
1170   - name: u
1171     histogram: {'00': 15, '01': 20, '10': 36, '11': 29}
1172     mitigation_info:
1173       target: {prob_meas0_prep1: 0.059000000000000005,
1174               prob_meas1_prep0: 0.0202}
1175       ancilla: {prob_meas0_prep1: 0.075400000000000002,
1176               prob_meas1_prep0: 0.0528}
1177       mitigated_histogram: {'00': 0.1187719606657805, '01':
1178                           0.1962085394489247, '10': 0.3710195249988589, '11':
1179                           0.31399997488643583}
1180

```

1181 For the same experiment file, we use the flag `asynchronous: true` to define
1182 asynchronous experiment.

Listing 17: Backend file

```

1183
1184 name: ibmq_quito
1185 asynchronous: true
1186 provider:
1187   hub: ibm-q
1188   group: open
1189   project: main
1190

```

1191 If the backend is asynchronous, the output will contain intermediate data

1192 containing, amongst others, job ids correlated with the circuit they corre-
 1193 spond to.

Listing 18: Resolved results

```

1194 metadata:
1195   experiments:
1196     type: discrimination-fourier
1197     qubits:
1198       - {target: 0, ancilla: 1}
1199       - {target: 1, ancilla: 2}
1200     angles: {start: 0.0, stop: 6.283185307179586, num_steps: 3}
1201     gateset: ibmq
1202     method: direct_sum
1203     num_shots: 100
1204   backend_description:
1205     name: ibmq_quito
1206     asynchronous: true
1207     provider: {group: open, hub: ibm-q, project: main}
1208 data:
1209   - job_id: 63e7f17a17b7ed49ca24e05b
1210     keys:
1211       - [0, 1, id, 0.0]
1212       - [0, 1, u, 0.0]
1213       - [0, 1, id, 3.141592653589793]
1214       - [0, 1, u, 3.141592653589793]
1215       - [0, 1, id, 6.283185307179586]
1216       - [0, 1, u, 6.283185307179586]
1217       - [1, 2, id, 0.0]
1218       - [1, 2, u, 0.0]
1219       - [1, 2, id, 3.141592653589793]
1220       - [1, 2, u, 3.141592653589793]
1221       - [1, 2, id, 6.283185307179586]
1222       - [1, 2, u, 6.283185307179586]
1223       - [1, 2, id, 6.283185307179586]
1224

```

1225 Finally, if the status of jobs is DONE, we resolve the measurements from the
 1226 submitted jobs obtaining the following file.

Listing 19: Results (asynchronous)

```

1227 metadata:
1228   experiments:
1229     type: discrimination-fourier
1230     qubits:
1231       - target: 0
1232

```

```

1233     ancilla: 1
1234 - target: 1
1235     ancilla: 2
1236 angles:
1237     start: 0.0
1238     stop: 6.283185307179586
1239     num_steps: 3
1240 gateset: ibmq
1241 method: direct_sum
1242 num_shots: 100
1243 backend_description:
1244     name: ibmq_quito
1245     asynchronous: true
1246     provider:
1247         group: open
1248         hub: ibm-q
1249         project: main
1250 data:
1251 - target: 0
1252     ancilla: 1
1253     phi: 0.0
1254     results_per_circuit:
1255 - name: id
1256     histogram:
1257         '00': 27
1258         '01': 28
1259         '10': 18
1260         '11': 27
1261     mitigation_info:
1262         target:
1263             prob_meas0_prep1: 0.052200000000000024
1264             prob_meas1_prep0: 0.0172
1265         ancilla:
1266             prob_meas0_prep1: 0.059000000000000005
1267             prob_meas1_prep0: 0.0202
1268     mitigated_histogram:
1269         '00': 0.254196166145997
1270         '01': 0.2790358060520916
1271         '10': 0.1732699847244092
1272         '11': 0.29349804307750227
1273 - name: u
1274     histogram:
1275         '00': 29

```

```

1276         '01': 17
1277         '10': 30
1278         '11': 24
1279     mitigation_info:
1280         target:
1281             prob_meas0_prep1: 0.0522000000000000024
1282             prob_meas1_prep0: 0.0172
1283         ancilla:
1284             prob_meas0_prep1: 0.0590000000000000005
1285             prob_meas1_prep0: 0.0202
1286     mitigated_histogram:
1287         '00': 0.2733793468261183
1288         '01': 0.1621115306717096
1289         '10': 0.3045273800167787
1290         '11': 0.2599817424853933
1291 - target: 0
1292     ancilla: 1
1293     phi: 3.141592653589793
1294     results_per_circuit:
1295     - name: id
1296         histogram:
1297             '00': 3
1298             '01': 5
1299             '10': 37
1300             '11': 55
1301     mitigation_info:
1302         target:
1303             prob_meas0_prep1: 0.05220000000000000024
1304             prob_meas1_prep0: 0.0172
1305         ancilla:
1306             prob_meas0_prep1: 0.0590000000000000005
1307             prob_meas1_prep0: 0.0202
1308     mitigated_histogram:
1309         '00': 0.006189545789708441
1310         '01': 0.016616709640352317
1311         '10': 0.3675478279476653
1312         '11': 0.6096459166222741
1313 - name: u
1314     histogram:
1315         '00': 56
1316         '01': 42
1317         '10': 2
1318     mitigation_info:

```

```

1319     target:
1320         prob_meas0_prep1: 0.052200000000000024
1321         prob_meas1_prep0: 0.0172
1322     ancilla:
1323         prob_meas0_prep1: 0.059000000000000005
1324         prob_meas1_prep0: 0.0202
1325     mitigated_histogram:
1326         '00': 0.55731929321128
1327         '01': 0.43367489257574243
1328         '10': 0.009005814212977551
1329 - target: 0
1330   ancilla: 1
1331   phi: 6.283185307179586
1332   results_per_circuit:
1333   - name: id
1334     histogram:
1335         '00': 18
1336         '01': 28
1337         '10': 30
1338         '11': 24
1339     mitigation_info:
1340         target:
1341             prob_meas0_prep1: 0.052200000000000024
1342             prob_meas1_prep0: 0.0172
1343         ancilla:
1344             prob_meas0_prep1: 0.059000000000000005
1345             prob_meas1_prep0: 0.0202
1346         mitigated_histogram:
1347             '00': 0.15258295844557557
1348             '01': 0.2829079190522524
1349             '10': 0.3071204587046501
1350             '11': 0.25738866379752195
1351   - name: u
1352     histogram:
1353         '00': 32
1354         '01': 28
1355         '10': 23
1356         '11': 17
1357     mitigation_info:
1358         target:
1359             prob_meas0_prep1: 0.052200000000000024
1360             prob_meas1_prep0: 0.0172
1361         ancilla:

```

```

1362     prob_meas0_prep1: 0.059000000000000005
1363     prob_meas1_prep0: 0.0202
1364     mitigated_histogram:
1365         '00': 0.3026150836796529
1366         '01': 0.28491749668524724
1367         '10': 0.23230862145681827
1368         '11': 0.18015879817828173
1369 - target: 1
1370     ancilla: 2
1371     phi: 0.0
1372     results_per_circuit:
1373     - name: id
1374       histogram:
1375         '00': 27
1376         '01': 16
1377         '10': 30
1378         '11': 27
1379     mitigation_info:
1380     target:
1381         prob_meas0_prep1: 0.059000000000000005
1382         prob_meas1_prep0: 0.0202
1383     ancilla:
1384         prob_meas0_prep1: 0.075400000000000002
1385         prob_meas1_prep0: 0.0528
1386     mitigated_histogram:
1387         '00': 0.256742095057232
1388         '01': 0.15000257115061383
1389         '10': 0.29821012040758116
1390         '11': 0.29504521338457296
1391 - name: u
1392     histogram:
1393         '00': 34
1394         '01': 22
1395         '10': 25
1396         '11': 19
1397     mitigation_info:
1398     target:
1399         prob_meas0_prep1: 0.059000000000000005
1400         prob_meas1_prep0: 0.0202
1401     ancilla:
1402         prob_meas0_prep1: 0.075400000000000002
1403         prob_meas1_prep0: 0.0528
1404     mitigated_histogram:

```

```

1405         '00': 0.3325088211394024
1406         '01': 0.22335261496979697
1407         '10': 0.2441636375921354
1408         '11': 0.19997492629866526
1409 - target: 1
1410   ancilla: 2
1411   phi: 3.141592653589793
1412   results_per_circuit:
1413   - name: id
1414     histogram:
1415       '00': 3
1416       '01': 9
1417       '10': 51
1418       '11': 37
1419   mitigation_info:
1420     target:
1421       prob_meas0_prep1: 0.059000000000000005
1422       prob_meas1_prep0: 0.0202
1423     ancilla:
1424       prob_meas0_prep1: 0.075400000000000002
1425       prob_meas1_prep0: 0.0528
1426   mitigated_histogram:
1427     '00': -0.016627023111853642
1428     '01': 0.06778554570877951
1429     '10': 0.53899887367658
1430     '11': 0.40984260372649417
1431 - name: u
1432   histogram:
1433     '00': 43
1434     '01': 45
1435     '10': 7
1436     '11': 5
1437   mitigation_info:
1438     target:
1439       prob_meas0_prep1: 0.059000000000000005
1440       prob_meas1_prep0: 0.0202
1441     ancilla:
1442       prob_meas0_prep1: 0.075400000000000002
1443       prob_meas1_prep0: 0.0528
1444   mitigated_histogram:
1445     '00': 0.42955729968594086
1446     '01': 0.49336080079582095
1447     '10': 0.04937406434533623

```



```

1448         '11': 0.02770783517290191
1449 - target: 1
1450   ancilla: 2
1451   phi: 6.283185307179586
1452   results_per_circuit:
1453   - name: id
1454     histogram:
1455       '00': 22
1456       '01': 19
1457       '10': 35
1458       '11': 24
1459   mitigation_info:
1460     target:
1461       prob_meas0_prep1: 0.059000000000000005
1462       prob_meas1_prep0: 0.0202
1463     ancilla:
1464       prob_meas0_prep1: 0.075400000000000002
1465       prob_meas1_prep0: 0.0528
1466   mitigated_histogram:
1467     '00': 0.19592641048040849
1468     '01': 0.18787721420415215
1469     '10': 0.3590258049844047
1470     '11': 0.25717057033103463
1471 - name: u
1472   histogram:
1473     '00': 27
1474     '01': 24
1475     '10': 25
1476     '11': 24
1477   mitigation_info:
1478     target:
1479       prob_meas0_prep1: 0.059000000000000005
1480       prob_meas1_prep0: 0.0202
1481     ancilla:
1482       prob_meas0_prep1: 0.075400000000000002
1483       prob_meas1_prep0: 0.0528
1484   mitigated_histogram:
1485     '00': 0.25555866817587225
1486     '01': 0.2429501641251142
1487     '10': 0.24509293912212946
1488     '11': 0.2563982285768841
1489

```
