Evaluating Calibration-Based Digital Twins for IBM Quantum Hardware Simulation

Edgars Bautra[1,*,†], Maksims Dimitrijevs[1,*,‡] and Abuzer Yakaryılmaz[1,§]

[1]*Center for Quantum Computing Science, University of Latvia, Latvia*

Abstract

The availability of quantum computers remains limited despite recent advancements in the field. Simulator configurations that use noise models built from device-specific calibration information could help compensate for this restricted availability, provided that the effects of noise can be reproduced accurately. In this work, We evaluate calibration-based digital twins for *IBM Quantum* hardware, aiming to reproduce hardware measurement outcomes on classical simulators. A workflow is presented that builds twins from downloadable calibration CSV files by mapping coherence times, gate and readout error rates, and operation durations to thermal-relaxation, depolarizing, and readout error channels, while reconstructing a directed coupling map to restore connectivity constraints during transpilation. Four twin variants are compared (CSV-built, backend-derived simulator, backend-derived noise model, and fake-backend snapshots) under a common execution and validation protocol. Experiments on two *IBM* QPUs, *ibm_brisbane* and *ibm_sherbrooke*, use randomized five-qubit circuits of depths 10, 20, and 30 across four optimization levels. *Weighted Jaccard Similarity* indicates that twins constructed from downloadable calibration CSV data often achieved the closest agreement with hardware, while backend-derived twins provided competitive and practical baselines. The results further show that agreement depends on both the target device and the transpilation settings, underscoring the need to validate digital twins for the specific execution setup rather than assuming transferability across devices.

Keywords

Quantum computers, digital twin, noisy simulation, Qiskit

1. Introduction

In today's world, quantum computers as a technology are undergoing rapid development. Although there are large companies, such as *IBM*, that offer access to these computers through their cloud services [1], demand still exceeds availability, leading to long queues of submitted circuits, because of which certain restrictions have been introduced, such as limited available computational time [2].

Quantum computer simulators can be of considerable help with this situation, allowing anyone to create and run circuits on everyday devices – classical computers, based on recreated properties of quantum physics [3]. In addition, less computational resources of quantum computers are being used by reducing the workload from unnecessary jobs that can be completed on simulators. Even so, there is one characteristic property of quantum computers that can cause the expected results to change in unpredictable ways – noise, including decoherence, gate errors, and readout errors [4], which makes ideal simulation insufficient. This creates a need for calibration-based digital twins of quantum computers, i.e., simulator configurations that use noise models built from device-specific calibration information to approximate the noisy behavior of a corresponding real device. Availability of digital twins can compensate for the restricted availability of real Quantum Processing Units (QPUs), as well as allow one to do preliminary testing before running jobs on QPUs, thus sparing quantum computational resources.

Baltic DB&IS 2026 Conference Forum and Doctoral Consortium, 28 June - 1 July 2026, Tartu, Estonia

*Corresponding authors.

† Main contributor to the paper, including the designing and performing of practical experiments.

‡ Advising the research, structuring the paper and researching for the background and related work.

§ Helping with the structure, factual integrity, missing parts, improving grammar and clarity of the paper.

edgars.bautra12@gmail.com (E. Bautra); maksims.dimitrijevs@lu.lv (M. Dimitrijevs); abuzer.yakaryilmaz@lu.lv (A. Yakaryılmaz)

0009-0002-3632-5897 (E. Bautra); 0000-0002-4225-7889 (M. Dimitrijevs); 0000-0002-2372-252X (A. Yakaryılmaz)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-06-09

CEUR-WS.org/Vol-4211/paper10.pdf

Recent research explicitly frames calibration-based emulation as a digital twin problem. In particular, calibration-driven emulation of superconducting transmon devices has been presented as a way to build a digital twin from calibration data and additional benchmarking information, showing that hardware-aware emulation can improve agreement with the behavior of a real device [5]. At the same time, *Qiskit* already provides practical support for noisy simulation based on backends, and *IBM* makes QPU calibration data downloadable as CSV [6, 7]. It is important to note that *Qiskit* can automatically configure a simulator to mimic an available backend via its specific methods, such as `AerSimulator.from_backend()`, while constructing a twin directly from downloadable calibration CSV files is not provided as a native end-to-end workflow in *Qiskit*. CSV files open up the possibilities to work with digital twins based on a wider range of QPUs, including ones not available to the user.

Calibration CSV data derived from *IBM* has been used in quantum digital twin and simulation contexts. For example, Luo et al. imported calibration data of *IBM* QPUs collected as CSV files, and extended the twin concept to quantum-cloud operations like job lifecycle and fidelity estimation [8]. Simulation frameworks such as *iQuantum* support importing *IBM*-style calibration CSV datasets for modeling quantum computing environments [9]. These studies show the value of calibration-aware simulation; however, to the authors' knowledge, they do not describe a practical, *Qiskit*-native workflow for building circuit-executable twins directly from downloadable *IBM* calibration CSV files and validating them against *IBM* hardware measurement distributions. This leaves a gap for users seeking simulators that capture both device noise and execution constraints.

We address this gap by developing and validating calibration-based digital twins of *IBM Quantum* hardware using *Qiskit* and downloadable *IBM* calibration CSV data. Calibration attributes are used to build noise models and to reconstruct the directional coupling information needed for a device-faithful simulation. This CSV-based approach is compared with backend-derived and snapshot-based alternatives, as well as all twin variants are validated against real *IBM* QPU outputs using *Weighted Jaccard Similarity*. The results show that twins based on CSV are feasible and often highly competitive, while backend-derived twins remain strong practical baselines.

The main contributions of this paper are as follows:

- A practical method is presented for constructing a QPU digital twin from downloadable *IBM* calibration CSV data, including the use of calibration attributes for noise-model creation and a workaround for reconstructing directional coupling information required for device-faithful transpilation and simulation.
- Four digital twin variants based on different sources of device and noise information (custom noise models built from downloadable calibration CSV files, backend-derived simulator and noise-model variants, and fake-backend snapshots) are formalized and compared.
- A validation procedure based on *Weighted Jaccard Similarity* is introduced for comparing measured state-count outputs from QPU runs and simulator runs under a common evaluation protocol.
- The impact of shot counts on result stability was investigated, thus, a practical shot budget was selected for calibration-based digital twin evaluation.
- The authors provide an empirical validation study on two *IBM Quantum* QPUs, using three randomly generated five-qubit circuits and four transpiler optimization levels to compare hardware results with simulator results across all twin variants.

The remainder of the paper is organized as follows. Section 2 describes the technical details of the conducted research. Section 3 presents the results and discussion, including implications, and limitations. Section 4 is the conclusion section.

The code, results of conducted experiments, and device calibration data files are publicly available at https://github.com/BautraE/precise-qpu-simulation-extras.

2. Technical details

2.1. Problem setting and digital twin definition

Local simulation provides the ability to compensate for constraints on access to real quantum hardware. When preparing and testing circuits before execution on a real QPU, the usefulness of local simulation depends on how closely the simulator reproduces the noisy behavior of the QPU.

In this paper, the problem is formulated as a *digital twin validation* problem. The output similarity between real-device runs and simulator runs produced by multiple twin variants is evaluated (agreement of measured circuit-execution results under noisy conditions).

The authors define *calibration-based digital twin* of a QPU as a simulator instance parameterized with device information in order to approximate the noisy execution behavior of that device. The digital twin consists of: a simulator backend configuration; a noise model; device execution constraints (e.g., basis gates and coupling topology) relevant for transpilation and simulation; an execution workflow that returns measurement counts comparable to QPU runs.

2.2. Validation goal and acceptance criteria

The main goal is to validate whether the twin is useful for reproducing circuit outputs observed for a QPU, i.e., whether the received results from a circuit execution are similar enough between a QPU and a simulator that is based on it. The comparison is performed at the level of measured state-count outputs, using a similarity metric defined in Section 2.9.1.

Author definitions of similarity thresholds for digital twins:

- 95% as near-identical to QPU;
- 90% as close match to QPU;
- 85% as usefully similar to QPU.

95% acceptance threshold is not uncommon, for example, in [10], authors set acceptance threshold of 0.95 for simulation fidelity. Thresholds of 90% and 85% have reasonable total variation distance ($\leq 5.26\%$ and $\leq 8.11\%$, respectively).

2.3. Twin variants and execution workflow

This subsection summarizes the digital-twin variants compared in the study and the common workflow used to construct, execute, and validate them. Four twin variants are evaluated within one common framework. This allows the differences in output agreement to be attributed to the twin-construction approach rather than to changes in the comparison procedure.

The evaluated digital twin variants are:

- Custom noise models from device calibration data (created from downloadable CSV files);
- Automatically obtained simulator instances that are based on a specific *IBM Quantum* QPU;
- Automatically obtained noise model instances that are based on a specific *IBM Quantum* QPU. It is closely related to the simulator instance (previous point), but not identical – the simulator variant also carries backend execution constraints such as basis gates and coupling map;
- Automatically obtained fake backends based on QPU system snapshots – *FakeBrisbane* and *FakeSherbrooke*.

The workflow has four main stages:

1. **Device-data acquisition and preparation.** Device-related information is collected from the target QPUs (basis gates and calibration data in the form of CSV files).
2. **Twin construction.** Instantiating simulators involves both manually creating new simulators and obtaining already prepared ones from *IBM*. Created simulators are then configured with noise models that are either created or simply obtained.

3. **Circuit execution.** The selected benchmark circuits are transpiled and executed on the real QPU and each twin variant.
4. **Validation comparison.** Similarity matrices and summary statistics for evaluation are produced based on measurement count outputs.

2.4. Calibration data extraction and preprocessing

Even though it is currently possible to simulate quantum computers and create noise models that mimic the effects of decoherence on classical computers, specific attribute values must be used when defining certain types of noise in order for the simulator results to align with those of real QPUs. In regard to this, *IBM* regularly conducts measurements and calculations to produce calibration data for every single qubit of their QPUs. During preprocessing, all time-valued attributes are converted to a common unit before being used in noise-parameter calculations. In particular, coherence times are reported in microseconds, while operation durations (readout and gate lengths) are reported in nanoseconds. The following attributes are available in the downloadable calibration data CSV files from the *IBM Quantum* platform, and their names are taken directly from these files [7]:

- **Qubit identification and status attributes:**
  - `Qubit` – index of qubit to which the calibration values apply;
  - `Operational` – boolean value that shows if the qubit is currently usable.
- **Coherence time attributes** are used for thermal relaxation noise:
  - `T1 (us)` – energy-relaxation time;
  - `T2 (us)` – phase-coherence time.
- **Control attributes** are relevant to hardware control and pulse-level behavior, but they are not used in our current gate-level noise-model construction:
  - `Frequency (GHz)` – each qubit has its own frequency that needs to be used when applying a certain gate;
  - `Anharmonicity (GHz)` – "the difference in energy between the first and second excited states of the qubit" [7].
- **Readout attributes:**
  - `Readout assignment error` – the probability to measure the wrong value for a qubit;
  - `Prob meas0 prep1` (`Prob meas1 prep0`) – probability that the measurement of a qubit's value returns '0' ('1') immediately after it was prepared as state $|1\rangle$ ($|0\rangle$).
- **Operation timing attributes** – these values are used with T1/T2 for thermal relaxation errors:
  - `Readout length (ns)` – measurement duration;
  - `Single-qubit gate length (ns)` – duration of a single-qubit gate;
  - `Gate length (ns)` – duration of a two-qubit gate on a qubit pair (in practice, it is direction-dependent).
- **Single-qubit depolarizing error attributes** are used for depolarizing errors:
  - `ID error`
  - `Z-axis rotation (rz) error`
  - $\sqrt{x}$ `(sx) error`
  - `Pauli-X error`
  - `RX error`
- **Two-qubit depolarizing error attributes** are used for depolarizing errors on qubit pairs (also direction-dependent):
  - `ECR error`

- CZ error
- RZZ error

Although only two QPUs (*ibm_brisbane* and *ibm_sherbrooke*) were available for free plan users while conducting practical tests for this paper, it was (and still is) possible to download QPU data from all active devices at no additional cost. With this in mind, if a QPU can be accurately simulated based on its calibration data, any such device can be replicated locally on classical computers free of charge, though existing restrictions of quantum computer simulation should be kept in mind, for example, extensive RAM usage. The approach may be adaptable to non-*IBM* QPUs with sufficiently similar exposed calibration and target information; evaluating that portability is left for future work.

2.5. Noise model construction from calibration data

Since the calibration data originates from the *IBM Quantum* platform, *IBM Qiskit* and *Qiskit Aer* are used to construct the simulators and noise models through functionality provided by specific classes and functions.

In the pipeline, `NoiseModel` stores created errors in a single object, `QuantumError` describes gate-related CPTP errors, and `ReadoutError` describes classical readout errors. The following helper functions are used to instantiate `QuantumError` objects: `depolarizing_error()` and `thermal_relaxation_error()`. `ReadoutError` is instantiated directly from the measurement-assignment probabilities. The calibration fields defined in Section 2.4 are used to obtain error objects that are then inserted into the `NoiseModel` object with `add_quantum_error()` and `add_readout_error()` for the corresponding qubits and qubit pairs..

The choice of these specific error classes and helper functions is supported by the official *Qiskit Aer* documentation, which describes how backend-derived noise is constructed in similar situations. Additionally, the `NoiseModel` class has a method called `from_backend()` that is used to obtain automatically created noise models from backend calibration properties. Its documentation also mentions the use of the previously described helper functions and classes.

Before creating any errors, a list of QPU-specific basis gates is passed as an argument to the `NoiseModel` object, because default settings may add unnecessary gates that cannot be removed afterwards.

2.6. Coupling map reconstruction from calibration tables

Each QPU also has its own specific coupling map that defines how qubits are connected. Furthermore, *IBM's* QPUs have directional coupling maps – not all directions between connected pairs are valid. This property is also essential for precise simulation; however, neither the neighboring qubits nor the coupling direction is directly mentioned anywhere in the calibration data. Though there is a workaround – there are usually multiple values stored in every qubit pair-related attribute column, and each of these values has a target qubit mentioned. Thus, it is possible to obtain the coupled qubit pairs by combining the current qubit (the control qubit) and the target qubits from these columns – though it takes additional effort.

Once the coupling topology is obtained, it can be defined with the `CouplingMap` class as a directed graph of permitted couplings. Since *NoiseModel* objects do not contain coupling maps, they are added to a simulator instance along with the created noise model (e.g., via `AerSimulator(..., noise\_model=..., coupling\_map=...)`).

2.7. Backend- and snapshot-based noise models

In addition to the possibility of creating custom noise models, *Qiskit* also provides functionality for obtaining automatically created noise models and simulators. Both `AerSimulator` and `NoiseModel` classes have the `from_backend()` method that configures the respective objects to mimic a target backend, similarly to the previously described pipeline in Sections 2.5 and 2.6.

The backend-derived construction depends on the backend availability under the current *IBM Quantum* access plan of the user. As an alternative, the module `qiskit_ibm_runtime.fake_provider` provides "fake backends" that mimic *IBM Quantum* systems using system snapshots. For those, the calibration data will not be up-to-date with the currently available QPUs on the platform. Additionally, snapshot coverage can vary across releases/devices (e.g., when the practical tests for this research were conducted, only 7 of 12 QPUs were available as fake backends).

2.8. Simulator configuration choices

It is possible to select one of multiple simulation methods as part of the simulator configuration; however, not all of them support measurements. Additionally, there are methods that can only be used with a GPU, but this functionality does not have support across all operating systems, so they were ignored.

Without going into depth on the differences of each method, a simple test was conducted to see if it is even worth testing out every valid simulation method separately. The three methods that met the previously stated constraints (`statevector`, `density_matrix`, `matrix_product_state`) were used to run all three circuits with 100000 shots, and their result similarities were compared. In all cases, the similarity between results from each simulation method was around 97.6%. Thus, it can be concluded that, at least in terms of results, there are no noticeable differences between the simulation methods.

As for the choice of `density_matrix` — it was the fastest of the three tested simulation methods.

2.9. Evaluation protocol

This subsection describes the protocol used to evaluate the agreement between QPU executions and the corresponding digital-twin executions. The evaluation is performed at the level of measured state-count outputs. The protocol specifies the similarity metric, the benchmark circuits, the transpilation settings, the shot-budget choice, and the practical data-collection setup used in the experiments.

2.9.1. Similarity metric

Since algorithms are typically executed multiple times on quantum computers, which is specified by the number of shots, the results for circuit execution jobs consist of specific state measurement counts, for example:

$$[0 : 531; 1 : 469]$$

In this example, '0' and '1' represent measured quantum states, whereas '531' and '469' are specific state count values – the numbers of times each of the states was measured as a result.

To obtain a specific percentage value that defines the similarity of such results, a custom function was created based on the *Weighted Jaccard Similarity* formula: [11]

$$J_w(x, y) = \frac{\sum_{i=1}^{n} \min(x_i, y_i)}{\sum_{i=1}^{n} \max(x_i, y_i)}$$

In simple terms, for each pair of comparable results, the function goes through all measured states and calculates the sums of *min* and *max* state counts from both results. After dividing these two sums accordingly, based on the previously presented *Weighted Jaccard Similarity* formula, and multiplying the result by '100', the required percentage values are obtained. The function would then return a similarity matrix that consists of the calculated result similarity percentage values. Below is an example of a returned similarity matrix:

$$\begin{bmatrix} 100.0 & 67.306.. \\ 67.306.. & 100.0 \end{bmatrix}$$

This approach makes it very easy to read the similarity of the tested results, as *Weighted Jaccard* provides a normalized and easily interpretable measure of overlap between two count distributions. By directly comparing the frequencies of shared outcomes, it gives a practical similarity score for

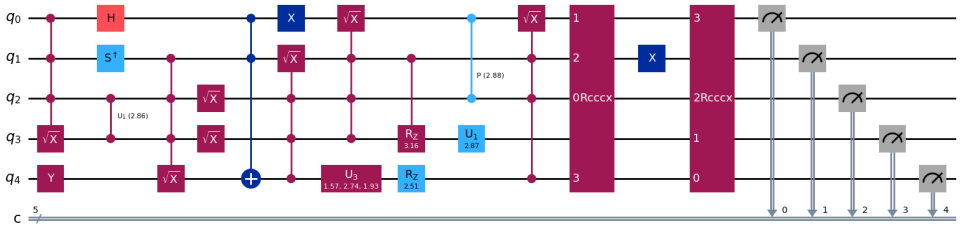

Figure 1: Generated five-qubit quantum circuit with depth 10.

simulator–hardware validation. As shown in the example, all diagonal values of the matrix are 100, which corresponds to each result being identical to itself. Each row of the matrix shows how one result instance (the one with the value '100') compares to all other result instances. Another characteristic of these similarity matrices is that values around the diagonal line will be mirrored — they will be the same on both sides of the diagonal line.

2.9.2. Benchmark circuits and transpilation settings

To obtain the required results, which could then be compared with the previously described method, there must be a quantum circuit that both the simulator and the real QPU will run.

It was decided to use three different randomly generated five-qubit quantum circuits — created by the *Qiskit* function `random_circuit()`, with circuit depths 10, 20, and 30. As the number of gates increases, the circuit becomes more susceptible to errors, which in turn allows the evaluation of a wider spectrum of different scenarios and their potential impact on the final results. Five-qubit quantum circuit with depth 10 that was used during the conducted practical tests is shown in Fig. 1.

Since the circuits are randomly generated, so are their expected results. A preliminary check was performed, comparing a noiseless simulation with a noisy simulation from a simulator based on one of the QPUs (obtained using the `from_backend()` method), to confirm that the chosen circuits are noise-sensitive. For example, the result difference for the same five-qubit circuit with depth 10 was 43,6%, confirming that this circuit is valid for further tests.

Validation practices in quantum computing often rely on characterization and benchmarks based on randomized circuits. This includes randomized benchmarking and randomized circuits of the *Quantum Volume* model [12, 13]. Algorithmic benchmarks like *Shor*-type circuits were avoided because these typically have more qubits and require longer execution times. For example, *Shor's* algorithm from a *Qiskit* tutorial [14] was initially tested, and attempts to execute on a real QPU ended with errors because the process took too long.

Before a quantum circuit can be run on any QPU or specific simulator instance, it must be transpiled accordingly. In simple terms, transpilation ensures required adjustments to the circuit:

- Mapping circuit qubits onto the coupling map of the QPU or simulator;
- Routing the circuit so that it is compatible with the coupling map of the QPU or simulator;
- Translating circuit gates to the basis gates that the QPU or simulator accepts.

When transpiling a circuit, it is also possible to optimize the process, which may help reduce the effects of noise and provide more precise results. However, it is important to note that these improvements are not always guaranteed or noticeable. Optimization can be configured to one of the four available levels, ranging from '0' to '3', which determines the type and extent of additional work done as part of the optimization process.

Since this would not deviate from the main goal of this research, but instead add additional potentially useful information in the form of test results, it was also decided to test whether different optimization levels would affect the similarity between simulator and QPU results. This meant that each of the three quantum circuits was tested with all four optimization levels.

Table 1
Obtained min and max similarity values from 100 circuit executions with specific shot counts

Shots	Circuit Depth 10		Circuit Depth 20		Circuit Depth 30	
	Min %	Max %	Min %	Max %	Min %	Max %
1000	72.265%	89.214%	73.611%	89.214%	75.131%	89.573%
5000	87.899%	95.274%	87.160%	95.007%	87.300%	95.160%
10000	91.296%	96.328%	91.314%	96.444%	90.949%	96.714%
15000	92.604%	97.109%	92.628%	97.381%	92.270%	96.992%
30000	94.723%	98.072%	94.514%	98.202%	94.779%	98.104%
60000	96.299%	98.603%	96.286%	98.606%	96.174%	98.613%
100000	97.022%	98.828%	96.984%	98.975%	96.817%	99.026%

2.9.3. Shot budget selection

The attribute `shots` determines how many times a quantum circuit is executed, and the statistical variability of circuit execution results decreases with increasing shot counts. Although higher shot counts typically yield more stable output distributions, it also leads to longer execution time, since quantum processing time scales with the number of shots. During preliminary research, a hard-coded limit of 100000 shots was discovered for both *ibm_brisbane* and *ibm_sherbrooke*, which can be obtained with `backend.configuration().max_shots`.

Even with this information, additional tests were conducted before picking a specific number to ensure the most optimal result similarity, as there was no guarantee that 100000 would be either enough or too excessive.

To answer this, all three randomly generated quantum circuits were each executed 100 times with an automatically generated noise model from a QPU and a specific shot count. The tests began with 1000 shots, then increased to 5000, after which all subsequent iterations would add 5000 more shots until reaching the previously mentioned hard-coded limit. During each iteration, all 100 obtained results were compared with one another, and the highest and lowest similarity values were recorded.

Table 1 contains some of the recorded lowest and highest result similarities for all three circuits with the optimization level set to '0' (full table is available at *GitHub*).

Based on the results, the most significant improvements occurred between 1000 and 15000 shots. Even though the rate of improvement decreased significantly at higher shot counts, improvement for both the highest and lowest similarity values was observable. Additionally, executing one circuit on a QPU with the maximum number of shots took approximately 40 seconds, which was considered an acceptable execution time. Since the hard-coded limit of 100000 shots continued to yield improvements in terms of obtained result similarity, along with the fact that the execution time per circuit was acceptable, there was no reason not to select it for further tests.

2.9.4. Hardware, data collection, and experimental matrix

The experiments were conducted on two *IBM Quantum* QPUs, which were available at the moment of conducting practical tests – *ibm_brisbane* and *ibm_sherbrooke*. For each device, three benchmark circuits were executed at four optimization levels and compared with four simulator-based twin variants, yielding the full experimental matrix used for validation. In total, 24 experiments were performed, each having one hardware job. For each hardware job, there are four simulator runs whose outcomes are compared with each other and the related hardware job (this makes 10 comparisons per experiment). As a result, there are 24 hardware jobs and 96 simulator runs.

During data collection, hardware jobs, saved noise data, and measurement counts were organized to reduce the effect of calibration drift over time. With the fact that it was only possible to submit three jobs to the queue simultaneously, along with potential waiting times spanning multiple hours, the jobs had to be divided accordingly so that there would be minimal impact of changing calibration data over

Table 2
Similarity matrix example for 1 of 24 experiments — for *ibm_brisbane*, five-qubit circuit with depth 10, and optimization level 3

	QPU	Simulator Instance From QPU	Noise Model Instance From QPU	*FakeBrisbane*	Noise Model From CSV
QPU	100%	75.940%	76.119%	70.136%	74.725%
Simulator Instance From QPU	75.940%	100%	98.078%	78.541%	95.114%
Noise Model Instance From QPU	76.119%	98.078%	100%	77.996%	95.532%
FakeBrisbane	70.136%	78.541%	77.996%	100%	75.634%
Noise Model From CSV	74.725%	95.114%	95.532%	75.634%	100%

Table 3
All 96 simulator results grouped into similarity intervals

Similarity Interval	Result Count			Percentage of Total Results			Cumulative Percentage		
	Total	bris.	sher.	Total	bris.	sher.	Total	bris.	sher.
Above 95%	7	7	0	7.29%	14.58%	0%	7.29%	14.58%	0%
Between 95% and 90%	17	7	10	17.71%	14.58%	20.83%	25.00%	29.17%	20.83%
Between 90% and 85%	25	13	12	26.04%	27.08%	25.00%	51.04%	56.25%	45.83%
Below 85%	47	21	26	48.96%	43.75%	54.17%	100%	100%	100%

time, which would otherwise lead to increased result differences. Thus, it was decided that for every set of three jobs, all three circuits would be submitted to run on a specific optimization level on one of the two devices.

Along with the instances of noise data, the result counts of each job were saved so that they could be later compared with the simulator results.

3. Results and discussion

3.1. Example similarity matrix and interpretation

The Table 2 shows a similarity matrix for one of the 24 tests as an example. The QPU results were compared with simulator results, each with one of the four selected noise data sources. In addition to this, simulator results could also be compared with one another.

3.2. Overall agreement distribution

Experiments produced a total of 96 simulator results to analyze (48 for each QPU). All results were grouped into similarity intervals for further analysis, which can be seen in the Table 3.

Based on the information presented in Table 3, the following observations can be made:

- Approximately half of all simulator results (51.04%) were with a similarity of 85% or higher, when compared to respective QPU results;
- Out of 96 simulator results, 7 exceeded the highest result similarity threshold of 95%. All 7 results were for the *IBM Quantum* QPU *ibm_brisbane*;
- If compared, there are more results with higher similarity for *ibm_brisbane* than there are for *ibm_sherbrooke*.

3.3. Which twin variant matches best

Table 4 highlights how often a specific noise data source produced the highest similarity value across all four tested sources, when compared to the QPU results.

Table 4
All 24 tests grouped by noise data source with the highest similarity to a QPU

Noise Data Source	Highest Similarity	Grouped by QPU	
		brisbane	sherbrooke
CSV calibration data	13 / 24	6	7
Noise model or simulator instance from backend (QPU)	7 / 24	5	2
Fake backends (*FakeBrisbane* and *FakeSherbrooke*)	4 / 24	1	3

3.4. Sensitivity to optimization level and device

There were some interesting recurring observations that are worth mentioning:

- When compared to other simulator results, instances that used fake backends (*FakeBrisbane* and *FakeSherbrooke*) as noise sources had noticeably lower similarity ratings if the optimization level was above 0;
- In general, optimization level changes affected the resulting similarity values in varying ways, making it difficult to draw definitive conclusions. The following examples illustrate this behavior:
  - Optimization level changes from 0 to 1 lead to increased similarity values. Though, at the same time, the similarity values decreased with level 2 and 3 (*ibm_brisbane* QPU, five-qubit circuit with depth 10);
  - Optimization level changes from 0 to 1 lead to decreased similarity values (*ibm_sherbrooke* QPU, same circuit as previous example).
- Backend-derived simulator and noise-model twins produced almost identical results, with the similarity consistently being around 95%. Furthermore, when comparing the two variant results to the QPU, both similarity values differed by only about 1%;
- In cases with the *ibm_brisbane* QPU, while comparing results from CSV calibration data-based twins to backend-derived simulator and noise-model variants, the similarity almost always exceeded 95%. In the very few cases where this did not occur, the similarity was still very close to 95%. However, this cannot be said about the *ibm_sherbrooke* QPU, as the similarity of the same twin variant results neither reached 95% nor was close enough to it;
- The highest result similarity values were always encountered with either optimization level 0 or 1. Only one exception occurred – running the five-qubit quantum circuit with depth 30 at optimization level 2 for *ibm_sherbrooke*.

3.5. Implications for calibration-based twins

The results indicate that CSV-based twins are a practical and often strong-performing option despite their higher implementation effort, while backend-derived twins remain useful baselines.

Twin similarity should be validated per device, since performance on one QPU does not guarantee comparable results on another. Moreover, results vary with transpilation settings, and so the transpiler configuration should be treated as part of the digital-twin definition and validation setup.

In several test conditions, simulator outputs from different twin variants are highly similar to each other while remaining noticeably less similar to the corresponding QPU output. This implies that high inter-simulator similarity is not a substitute for QPU validation.

The reported threshold outcomes should be interpreted alongside the full similarity distribution. For practical use, the required similarity level may depend on the intended application.

3.6. Threats to validity and limitations

The study used the platform for a maximum of 100000 shots. Higher shot counts might further improve agreement, but they were outside the practical limits of the experimental setup.

Even though it was not tested in the scope of this paper, circuits with different qubit counts could potentially give different results.

Along with this, the transpilation process may potentially play a role in determining the similarity. As was brought up in Section 2.9.2, optimization affects the types of changes done to a circuit before it finishes the transpilation process, and some changes might not be consistent when transpiling the same circuit multiple times with a higher optimization level (2 or 3). Because transpilation for the QPUs and simulators was done independently, there is a possibility that there were some differences that might have affected the final results.

Assumed noise model class has limitations, and more complex (possibly non-*Markovian*) effects can affect the difference between the outcomes of simulators and hardware [15]. Models based only on calibration can be improved by fitting parameterized models from circuit-execution data, potentially improving distribution-level agreement [16].

4. Conclusion

We show that calibration-based digital twins of *IBM Quantum* hardware are feasible under the tested validation protocol. Twins constructed from downloadable calibration CSV data were often the strongest performers, while backend-derived twins provided competitive and practical baselines. Our findings also show that agreement depends on the target device and on the transpilation settings, so validation should be carried out for the intended execution setup rather than being assumed to transfer automatically across devices. Our results are limited to the tested devices, circuits, shot budget, and noise-model assumptions.

Since the CSV-based noise models frequently demonstrated the highest agreement with hardware results, they represent a promising foundation for further refinement and optimization. Future studies could therefore focus specifically on improving calibration-driven CSV twin construction methods, with the aim of increasing the similarity between simulator and QPU results.

Future research should also extend the analysis to a broader range of circuit classes, qubit and shot counts in order to identify factors that might lead to improvements for the similarity of obtained results. Additionally, different transpilation settings should be systematically investigated, as the functionality provided by *IBM* Qiskit allows for variation that impacts the final form of the executable quantum circuit.

Acknowledgments

The work was supported by the Latvian Quantum Initiative under the European Union Recovery and Resilience Facility project no. 2.3.1.1.i.0/1/22/I/CFLA/001.

Declaration on generative AI

Over the course of preparing this work, the authors used *ChatGPT* for the following purposes: helping with paraphrasing and rewording specific sentences and parts of text; searching for related work and summarizing the contents; researching specific topics and finding valid sources of information. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] H. T. Nguyen, P. Krishnan, D. Krishnaswamy, M. Usman, R. Buyya, Quantum cloud computing: a review, open problems, and future directions, arXiv preprint arXiv:2404.11420 (2024). doi:10.48550/arXiv.2404.11420.

[2] J. Romero-Álvarez, J. Alvarado-Valiente, J. Casco-Seco, E. Moguel, J. Garcia-Alonso, J. M. Murillo, A noise validation for quantum circuit scheduling through a service-oriented architecture, International Journal of Software Engineering and Knowledge Engineering 34 (2024) 1371–1386. doi:10.1142/S0218194024410018.

[3] A. Cicero, M. A. Maleki, M. W. Azhar, A. F. Kockum, P. Trancoso, Simulation of quantum computers: Review and acceleration opportunities, ACM Transactions on Quantum Computing 7 (2025) 1–35. doi:10.1145/3762672.

[4] R. P. Antón, A. Corbi, J. I. L. Sánchez, D. Burgos, Reliability of IBM's public quantum computers., International Journal of Interactive Multimedia and Artificial Intelligence 9 (2025) 155–163. doi:10.9781/ijimai.2023.04.005.

[5] R. Müller, M. Zanner, M. Schielein, M. Rüfenacht, D. Rabanus, E. Schätzle, K. Michielsen, A. K. Karnad, D. Willsch, E. Jennings, et al., Towards a digital twin of noisy quantum computers: Calibration-driven emulation of transmon qubits, arXiv preprint arXiv:2504.08313 (2025). doi:10.48550/arXiv.2504.08313.

[6] IBM, Build noise models, 2025. URL: https://quantum.cloud.ibm.com/docs/en/guides/build-noise-models, last accessed 2026/03/09.

[7] IBM, View backend details, 2025. URL: https://quantum.cloud.ibm.com/docs/en/guides/qpu-information, last accessed 2026/03/09.

[8] W. Luo, B. Baheri, T. Humble, J. Zhao, T. Zhan, R. Maharjan, Q. Guan, A digital twin of scalable quantum clouds, in: Proceedings of the 39th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '25, Association for Computing Machinery, New York, NY, USA, 2025, p. 165–175. doi:10.1145/3726301.3732296.

[9] H. T. Nguyen, M. Usman, R. Buyya, iQuantum: A toolkit for modeling and simulation of quantum computing environments, Software: Practice and Experience (2024) 1–31. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3331. doi:https://doi.org/10.1002/spe.3331.

[10] O. Bertomeu, H. Ghayas, A. Roman, S. DiAdamo, Maestro: Intelligent execution for quantum circuit simulation, arXiv preprint arXiv:2512.04216 (2025). doi:10.48550/arXiv.2512.04216.

[11] A. Schubert, A. Telcs, A note on the jaccardized czekanowski similarity index, Scientometrics 98 (2014) 1397–1399. doi:10.1007/s11192-013-1044-2.

[12] E. Magesan, J. M. Gambetta, J. Emerson, Scalable and robust randomized benchmarking of quantum processes, Phys. Rev. Lett. 106 (2011) 180504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.106.180504. doi:10.1103/PhysRevLett.106.180504.

[13] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, J. M. Gambetta, Validating quantum computers using randomized model circuits, Phys. Rev. A 100 (2019) 032328. URL: https://link.aps.org/doi/10.1103/PhysRevA.100.032328. doi:10.1103/PhysRevA.100.032328.

[14] IBM, Shor's algorithm, 2025. URL: https://quantum.cloud.ibm.com/docs/en/tutorials/shors-algorithm, last accessed 2026/03/07.

[15] E. Onorati, T. Kohler, T. S. Cubitt, Fitting quantum noise models to tomography data, Quantum 7 (2023) 1197. URL: https://doi.org/10.22331/q-2023-12-05-1197. doi:10.22331/q-2023-12-05-1197.

[16] Y. Ji, M. Roth, D. A. Kreplin, I. Polian, F. K. Wilhelm, Data-efficient quantum noise modeling via machine learning, Physical Review Applied 25 (2025) 034051. doi:10.48550/arXiv.2509.12933.