Software Impacts 12 (2022) 100260

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Noise fingerprints in quantum computers: Machine learning software tools

Stefano Martina [a,b,*], Stefano Gherardini [d,e,b], Lorenzo Buffoni [c], Filippo Caruso [a,b]

[a] Department of Physics and Astronomy, University of Florence, Via Sansone 1, Sesto Fiorentino, I-50019, Italy
[b] European Laboratory for Non-Linear Spectroscopy (LENS), University of Florence, Via Nello Carrara 1, Sesto Fiorentino, I-50019, Italy
[c] Physics of Information and Quantum Technologies Group, Instituto de Telecomunicações, University of Lisbon, Av. Rovisco Pais, Lisbon, P-1049-001, Portugal
[d] CNR-INO, Area Science Park, Strada Statale 14, Basovizza (TS), I-34149, Italy
[e] Scuola Internazionale Superiore di Studi Avanzati (SISSA), Via Bonomea, 265, Trieste, I-34136, Italy

ARTICLE INFO

Keywords:
Machine learning (ML)
Support vector machines (SVM)
Quantum computers
Noisy Intermediate Scale Quantum (NISQ) algorithms
Qiskit
Scikit-learn

ABSTRACT

In this paper we present the high-level functionalities of a quantum–classical machine learning software, whose purpose is to learn the main features (the fingerprint) of quantum noise sources affecting a quantum device, as a quantum computer. Specifically, the software architecture is designed to classify successfully (more than 99% of accuracy) the noise fingerprints in different quantum devices with similar technical specifications, or distinct time-dependences of a noise fingerprint in single quantum machines.

Code metadata

Current code version	*v1*
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2022-9
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/8363708/tree/v1
Legal Code License	*GNU General Public License v3.0*
Code versioning system used	*GIT*
Software code languages, tools, and services used	*python, IBM quantum services*
Compilation requirements, operating environments & dependencies	numpy, qiskit, qiskit_terra, scikit_learn
If available Link to developer documentation/manual	
Support email for questions	stefano.martina@unifi.it

1. Introduction

Quantum technologies are a fast developing scientific and industrial field [1]. They have been already implemented in several different platforms, as for instance photonic circuits [2,3], but also Rydberg atoms [4], superconducting devices [5] and others. Likely, the most promising quantum technology is represented by quantum computers, i.e., quantum devices for quantum computing, among which it is worth mentioning superconducting circuits [6,7], trapped-ions quantum computers [8,9], photonic chips [10,11] and topological qubits [12]. Both academic laboratories and industrial companies are devoting lots of effort and funding to boost research and technological improvements, towards the so-called *quantum supremacy* [13], i.e., a quantum advantage to solve (numerical) problems that no classical computer will never solve. The actual drawback of these devices is the absence of a standard hardware (and thus even software) architecture, on which research activities may be jointly coordinated. For each of these platforms, indeed, ad hoc solutions are proposed and then realized, and this makes such a technologies still very expensive and incompatible from a device to another.

However, in quantum computing, the main issue to be still solved is the unavoidable presence of external noise sources that dramatically

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author at: Department of Physics and Astronomy, University of Florence, Via Sansone 1, Sesto Fiorentino, I-50019, Italy.
E-mail address: stefano.martina@unifi.it (S. Martina).

https://doi.org/10.1016/j.simpa.2022.100260
Received 5 February 2022; Accepted 12 February 2022

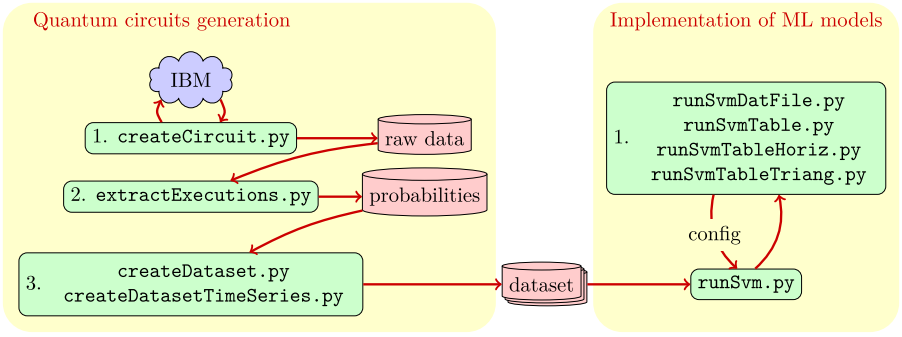

Fig. 1. Pictorial figure showing the structure of the software architecture and how its different parts depend each other. On the left, one can observe the part of the software that is designed for the generation of the testbed quantum circuit. Specifically, `createCircuit.py` is used to launch the quantum circuit on the IBM cloud services with the aim to get the raw data from the measurement procedure in each execution of the circuit. The file `extractExecutions.py` computes the probabilities to get the measurement outcomes from the execution of the testbed quantum circuit. Either `createDataset.py` and `createDatasetTimeSeries.py` is used to pack in datasets the measurement outcomes probabilities. The former creates datasets with data collected on two or more machines, while the latter collects data taken on the same machine but at different times. On the right side of the figure, we represent the workflow for the training of ML methods. Specifically, the files `runSvmDatFile.py`, `runSvmTable.py`, `runSvmTableHoriz.py` and `runSvmTableTriang.py` are employed to generate the output data that we have shown in the tables and plots in [17]. All these scripts call the function `runSVM` in the file `runSvm.py` that contains the main code for the definition, training and evaluation of the SVM models. Note that the function is called with one of the configuration names (`config` in the picture) that is listed in the config file `configurations.py`. The configuration name denotes what is the generated dataset that is used for the training of the ML models.

limit the accuracy of quantum computations, as well as the large-scale realization of quantum circuits and algorithms. The negative impact of noise on quantum computing is so noticeable that the acronym *Noisy Intermediate-Scale Quantum* (NISQ) technology has been recently introduced [14]. Furthermore, commercial quantum devices as for example the quantum computers by the companies Q-IBM® [15] and Rigetti® [16], albeit they have been made available by anyone who creates a free account on their database, are not physically accessible and several specifications on the chip's parameters are not made public.

In the paper [17], we have recently observed on some IBM quantum computers that main features of the noise sources affecting the devices are specific of each single computer and have a clear time-dependence. For such a purpose, a *testbed quantum circuit* – composed by a fixed number of qubits – is designed, then made run for a sufficient number of times and finally locally measured in correspondence of each qubit. From the measurements of the qubits (the measurement observable was the Pauli matrix σ_z), a set of measurement outcomes is recorded, collected, and then used to train a *machine learning* (ML) algorithm [18,19]. However, it is worth noting that in [17] the features of the quantum noise are not reconstructed but just classified from a quantum device to another. Specifically, the classification task was successfully carried out by means of a *support vector machine* (SVM) [18,19], with a classification accuracy equal or greater than 99%. Hence, thanks to our procedure, one just needs to collect an informative statistics of quantum measurement outcomes (that are *quantum data*) from the testbed quantum circuit, and subsequently train ML (classical) algorithms. In fact, no quantum noise modelling is required nor, in principle, the testbed circuit has to be controlled by time-dependent pulses [20]. Also for these reasons, the use of a ML technique is the most natural choice to perform classification, since it naturally provides a black-box model with predictive outcomes. In this regard, we recall that in the current literature ML has been already adopted to distinguish open quantum dynamics [21–23] and to perform quantum sensing tasks [24–27], as for example the learning and classification of non-Markovian noise [24,26] or the detection of qubits correlations [25].

As depicted in Fig. 1, our software architecture adopted in [17] has two distinct parts: The one on the left of the figure generates the testbed quantum circuit (see Section 2), while the other is designed for the implementation of the ML models that classify quantum noise fingerprints (refer to Section 3).

2. Testbed quantum circuit

For our experiments of quantum noise classification in [17], we made use of the IBM Quantum cloud services to remotely run quantum circuits on different machines. In particular, to interact with the cloud services, one can use the Qiskit sdk [28] that is an open-source python package, useful both to *simulate* quantum dynamics and to *program* a given set of operations on a real quantum computer. Currently, one has at disposal up to 11 superconducting quantum computers, ranging from a single qubit up to 15 qubits, with different topology and calibration routines. For all the available devices and their specifications, we direct the reader to the IBM documentation [15].

Algorithm 1: Generation of the testbed quantum circuit (baseline version)

Require: IBM-Q backend (specific device to fingerprint)
Ensure: $|0\rangle_i \ \forall i \in 0, ..., 3$
  for number of repetitions **do**
    $0 \leftarrow H$ ▷ Hadamard gate on the 0^{th} qubit
    $1 \leftarrow H$
    CNOT$(0 \rightarrow 2)$ ▷ Controlled NOT gate on the 2^{nd} qubit conditioned on the qubit 0
    CNOT$(1 \rightarrow 3)$
    $0 \leftarrow X$ ▷ X gate on the 0^{th} qubit
    $1 \leftarrow X$
    Toffoli$(0, 1 \rightarrow 2)$ ▷ Toffoli gate on the 2^{nd} qubit conditioned on the qubits $0, 1$
  end for
  Measure(2) ▷ Projective measurements of the i^{th} qubit
  Measure(3)
  return 1000 shots from the measurements

In Algorithm 1 we provide the pseudo-code for the generation of the testbed quantum circuit in its baseline version (see also panel (c) in Fig. 2 for a pictorial representation of the circuit) realized in [17] to carry out the classification of noise fingerprints. For the quantum computation in the aforementioned circuit, we made use of standard gates whose mathematical definitions is given in terms of matrices that one can easily find in quantum computing textbooks [29]. The baseline version of the testbed quantum circuit is repeated 3 times overall for a total of 9 measurements (also denoted as measurement steps) of both qubits 2 and 3. Indeed, operationally, the 9 measurement steps are not performed all in the same run (i.e., sequentially), but on consecutive runs by implementing incremental parts of the quantum circuit. To clarify this aspect as much as possible, in Fig. 2 we have represented pictorially the first 4 measurement steps of the testbed quantum circuit, whose baseline version (returned by Algorithm 1) is

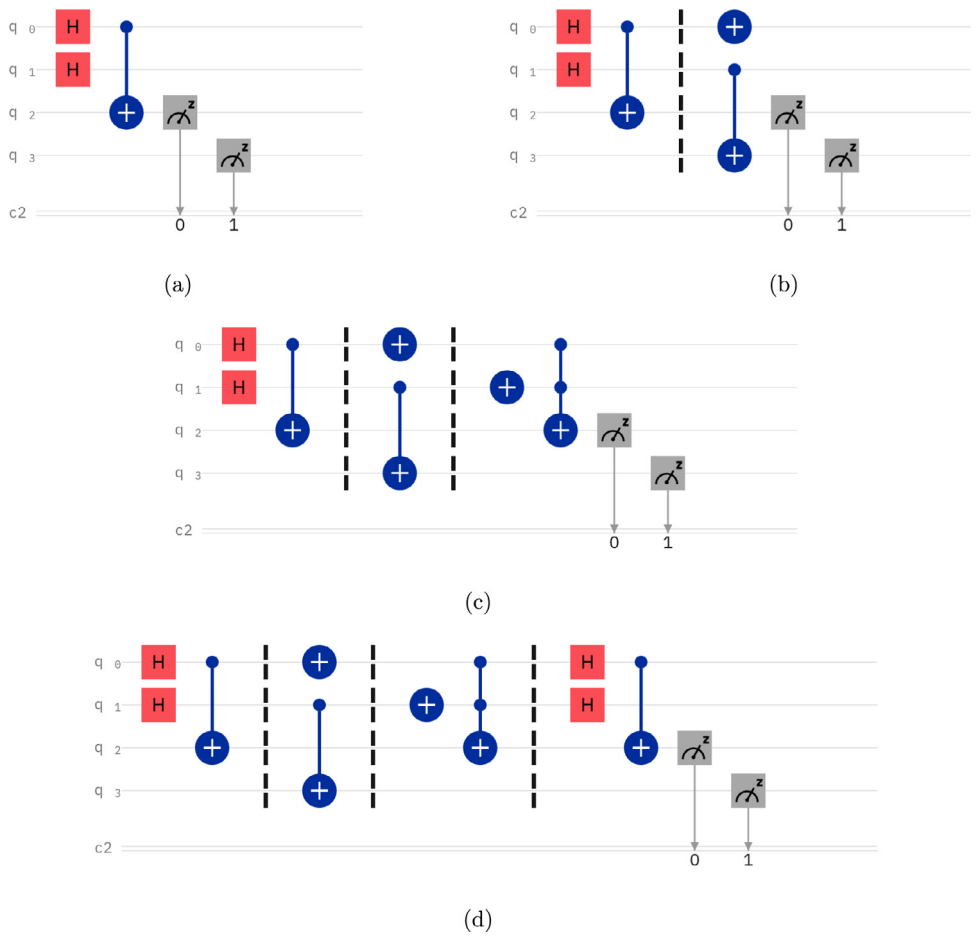

Fig. 2. Pictorial representation of the first 4 measurement steps (the panels of the figure) applied to the testbed quantum circuit designed in Ref. [17]. The 4 panels have to be read from left to right, and from top to bottom. The baseline version of the testbed quantum circuit (provided by Algorithm 1) is the one depicted in panel (c), in correspondence of the third measurement step. The full set of measurement outcomes is obtained by repeating 3 times the baseline circuit and then performing a total of 9 measurement steps, each of them acting on the qubits 2 and 3. The outcomes collected in the 9 measurement steps come from executing incrementally the testbed quantum circuit in different runs, where the qubits 2 and 3 are measured only at the end of the implemented circuits.

depicted in the panel (c) of the figure. Specifically, first we execute the part of the circuit that is obtained by cutting the testbed quantum circuit after the first measurement step, i.e., after the measurement of qubit 2 and 3 following the Hadamard gates on the 0th and 1th qubits and the CNOT gate from qubit 0 to 2). Then, the measurement outcomes are recorded. Subsequently, we execute part of the testbed quantum circuit until the second measurement step (measurements of qubits 2 and 3 included), thus by also taking into account the X-gate on the 0th qubit and the CNOT gate from qubit 1 to 3, and again we record the measurement outcomes. The procedure is then repeated for all the 9 measurement steps. Before proceeding, it is worth stressing that, for each implemented quantum circuit, the measurements of the qubits 2 and 3 are performed only at the end of the circuits.

Overall, in [17], several experiments have been conducted on different IBM chips that have different physical specifications, as the architecture of the qubits or the quantum volume [30]. Some quantum machines, indeed, are inherently noisier than other, and even single qubits inside a machine can have a distinctive noise profile. All these peculiar differences in noise and topology contribute to the fingerprint that we aim to classify using our ML method.

2.1. Data acquisition

The pipeline designed for the creation of the dataset, set as input of the ML models, is constituted of several scripts that can be customized according to the needs of the user. First, for each implemented quantum circuit, the script `createCircuit.py` adopts Qiskit to interact with the IBM quantum services for the measurement of a predefined number of circuit executions. Specifically, such a script is parameterized to launch the runs of the circuits on several quantum machines with a specific amount of parallel tasks. The runs are executed in two different modalities that generates the datasets that we called FAST and SLOW in [17]. In the former dataset, the aim was to collect as many runs as possible in the shortest time interval. For this purpose, the script launches 20 parallel processes, each of them adds to the IBM queue a predefined number of runs with 8000 execution-shots. After that, each batch of 8000 execution-shots is split into 8 batches of 1000 shots that are then employed to compute the outcomes' probabilities. Instead, for the dataset named SLOW, we collect a sequence of measurement outcomes that are uniformly distributed over time. To obtain such dataset, the script launches only one run at a time with 1000 execution-shots and waits two minutes from a run to another.

The second script in the pipeline is `extractExecutions.py`, whose objective is to compute the probabilities of the measurement outcomes from the raw data returned by the calls to the IBM quantum services (this is ensured by the previous script). After that, either `createDataset.py` and `createDatasetTimeSeries.py` pack the probability distributions in datasets. The difference between such scripts is the following. The former builds binary or multiclass classification dataset using data from at least two quantum machines, while the latter builds classification datasets with data collected in a single machine and labelled by the time interval in which the testbed quantum circuit is executed.

In the github repository at https://github.com/trianam/learningQuantumNoiseFingerprint and on CodeOcean at https://codeocean.com/capsule/fa6e1d85-c99f-4a38-9c16-ac204da85040/, we release the source code of all the scripts and all the data obtained from the execution of `createCircuit.py` on each quantum machine we employed.

3. Machine learning models

3.1. Support vector machine

In [17] we have successfully classified the noise fingerprints on several IBM quantum computers by training Support Vector Machine models [18,19]. SVM is a machine learning technique that is usually used to solve binary classification tasks. Generally speaking, a SVM model is trained on a dataset composed of pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ are points in a certain space $\mathbb{R}^n$ of dimension n, and y_i is equal to 1 or -1 depending the corresponding point belongs to one or the other class. The Support Vector Machine is trained to find the hyperplane that divides the space representation of the two classes, by ensuring the maximum distance from the points. When the points of the two classes are not linearly separable, a common solution is to resort to the so-called "kernel trick", i.e., the points $\mathbf{x}_i$ are mapped to a larger dimension space until they become linearly separable. The most common kernels are polynomial functions with varying degree number and the so-called Radial Basis Functions (RBF) [18,19]. Finally, SVM models can also be extended to multiclass classification tasks using the strategies One-Versus-All (OVA), or One-Versus-One (OVO) [18,19].

In our work, to implement and train the SVM models, we leveraged the *scikit-learn* python library [31].

3.2. Data interpretation

The code that implements and train the SVM is defined by the functions in the file `runSVM.py`. Specifically, the main function is called `runSVM`: it requires a configuration object that *(i)* identifies what model has to be used, and *(ii)* set optional arguments to tune the number of hyperparameters (`mask`) and to control if the method is verbose (`verbose`) and if the results have to be written in an output file (`writeToFile`). Practically, the function `runSVM` first calls `extractData` whose purpose is to load the dataset file, extract the data in the desired time steps and split them in *training*, *validation* and *test* sets. After that, `runSVM` proceeds to train a set of possible SVM models on the training set, by then evaluating them on the validation set and computing on the test set the resulting accuracy of the model that performed better on the validation set. The possible models that can be employed are: *(i)* Standard linear SVM (using two different libraries), *(ii)* SVM with *polynomial* kernel with degree from 2 to 4, and *(iii)* SVM with RBF kernel.

The file `runSVM.py` is provided with a `main` method. Thus, it can be directly called as a script using the configuration name as argument. We have also designed some useful methods that call `runSVM`, build directly the latex table with the results and calculate the points for the figures shown in [17].

4. Impacts

In this paper, we have explained in great detail the software architecture of the ML method, introduced in [17], to carry out quantum noise classification. Such a tools are intended to be applied to quantum technologies, as e.g., quantum computers.

The main impact of our software lies in its ability in classifying the fingerprint left by quantum noise sources on devices that have identical technical specifications and are thus expected to provide the same outcomes. Unfortunately, in quantum machines, the influence of the environment is so relevant that different noise fingerprints can be identified depending on the type of quantum computer (as previously explained, quantum computers can differ, e.g., on the number of qubits and/or the quantum volume), on the time period in which the single machine has worked, and on environmental changes mainly due to temperature fluctuations. However, thanks to our quantum–classical machine learning method, one can *(i)* distinguish the noise fingerprints in different quantum devices; *(ii)* classify the noise fingerprint on the same quantum devices but in different times; *(iii)* learn if and how a given noise fingerprint changes over time.

In [17] our method is proved to be very *accurate* (more than 99% of effectiveness) in classifying a clear machine-related noise fingerprint in each of the analysed IBM quantum computers, and even *robust* since any noise fingerprint is highly predictable over time in windows of consecutive runs. Also an evident time-dependence of the noise fingerprints has been classified, by observing changes over time after few hours from the first execution of the testbed quantum circuit.

Another important feature of our software architecture is that the ML models do not require a complete set of measurement outcomes as input data, but conversely the outcomes from a sequence of repeated measurements of a single observable. For an example, for the experiments in [17], the chosen observable was the tensor product of $sigma_z$ Pauli matrices locally applied on each qubit of the testbed circuit. Furthermore, the proposed method is able to distinguish and classify noise fingerprints, even without knowing the microscopic model that describes the (real or effective) interaction between the device and the external noise fields. This important aspect allows the user to employ our quantum–classical machine learning algorithm to classify the noise fingerprints of even *inaccessible* quantum machines.

4.1. Applications

The experimental evidences in [17] lead us to conclude that different quantum devices exhibit distinctive, and thus distinguishable, noise fingerprints that one can classify and predict. Therefore, in principle, our method could be adopted *to identify* from which specific quantum device certain data (a collection of measurement outcomes) are generated, just looking at the noise fingerprint of the device. Moreover, the proposed solution might be employed *to certify* the time-scheduling in which a given quantum computation is executed. Both these applications are expected to play a key role for diagnostics purposes – especially in all those contexts where quantum computations cannot be error-corrected [32] – and to accomplish benchmarking and certification [33,34] of quantum noise sources within a default error threshold.

4.2. Outlook

The proposed methodology, aimed to learn the noise fingerprint of quantum devices from time-ordered measurements of a testbed quantum circuit, may be in principle applied to any quantum devices, and thus not only to the IBM quantum computers as done in [17]. The possibility to predict on which device, and at which time, a given quantum operation (even time-varying) has been executed is expected to help the mitigation of quantum computational errors (e.g., by means of calibration routines), and to assist the application of ad-hoc error corrections.

Furthermore, instead of SVMs, one could employ deep learning techniques, as for example Recurrent Neural Networks (RNN) [18,35,36], to make more efficient the classification of quantum noise fingerprints. In such a case, the software architecture should be modified a bit, but not in the part that concerns the generation of the quantum data placed in input to the ML model. What should be different, indeed, is the way the input data would be processed.

Finally, we are also confident that, thanks to specific modifications, it is possible to carry out even the reconstruction of some quantum noise features. However, for such a purpose, a minimal knowledge of the way noise sources affect the quantum device under investigation will be required.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We acknowledge the access to advanced services provided by the IBM Quantum Researchers Program.

This work was financially supported from Fondazione CR Firenze, Italy through the project QUANTUM-AI, from University of Florence through the project Q-CODYCES, and from the European Union's Horizon 2020 research and innovation programme under FET-OPEN Grant Agreement No. 828946 (PATHOS).

References

[1] J.P. Dowling, G.J. Milburn, Quantum technology: the second quantum revolution, Philos. Trans. R. Soc. A 361 (2003) 1655–1674.
[2] J.L. O'Brien, A. Furusawa, J. Vučković, Photonic quantum technologies, Nat. Photon 3 (2009) 687–695.
[3] J. Wang, F. Sciarrino, A. Laing, M.G. Thompson, Integrated photonic quantum technologies, Nat. Photon 14 (2020) 273–284.
[4] C.S. Adams, J.D. Pritchard, J.P. Shaffer, Rydberg atom quantum technologies, J. Phys. B At. Mol. Opt. 53 (2020) 012002.
[5] Michel H. Devoret, Andreas Wallraff, John M. Martinis, Superconducting qubits: A short review, 2004, arXiv preprint Cond-Mat/0411174.
[6] John Clarke, Frank K. Wilhelm, Superconducting quantum bits, Nature 453 (7198) (2008) 1031–1042.
[7] Y. Wu, W. Bao, S. Cao, et al., Strong quantum computational advantage using a superconducting quantum processor, Phys. Rev. Lett. 127 (18) (2021) 180501.
[8] David J Wineland, Murray Barrett, Joseph Britton, J Chiaverini, B DeMarco, Wayne M Itano, B Jelenković, Christopher Langer, Dietrich Leibfried, V Meyer, et al., Quantum information processing with trapped ions, Phil. Trans. R. Soc. A 361 (1808) (2003) 1349–1361.
[9] I. Pogorelov, T. Feldker, Ch. D. Marciniak, L. Postler, G. Jacob, O. Krieglsteiner, V. Podlesnic, M. Meth, V. Negnevitsky, M. Stadler, B. Höfer, C. Wächter, K. Lakhmanskiy, R. Blatt, P. Schindler, T. Monz, Compact ion-trap quantum computing demonstrator, PRX Quantum 2 (2021) 020343.
[10] Justin B Spring, Benjamin J Metcalf, Peter C Humphreys, W Steven Kolthammer, Xian-Min Jin, Marco Barbieri, Animesh Datta, Nicholas Thomas-Peter, Nathan K Langford, Dmytro Kundys, et al., Boson sampling on a photonic chip, Science 339 (6121) (2013) 798–801.
[11] Benjamin J Metcalf, Justin B Spring, Peter C Humphreys, Nicholas Thomas-Peter, Marco Barbieri, W Steven Kolthammer, Xian-Min Jin, Nathan K Langford, Dmytro Kundys, James C Gates, et al., Quantum teleportation on a photonic chip, Nat. Photonics 8 (10) (2014) 770–774.
[12] Michael Freedman, Alexei Kitaev, Michael Larsen, Zhenghan Wang, Topological quantum computation, Bull. Amer. Math. Soc. 40 (1) (2003) 31–38.
[13] F. Arute, K. Arya, R. Babbush, et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (2019) 505–510.
[14] John Preskill, Quantum computing in the NISQ era and beyond, Quantum 2 (2018) 79.
[15] 2022, https://quantum-computing.ibm.com/. Visited on 2022.
[16] 2022, https://www.rigetti.com/. Visited on 2022.
[17] Stefano Martina, Lorenzo Buffoni, Stefano Gherardini, Filippo Caruso, Learning the noise fingerprint of quantum devices, 2021, arXiv preprint arXiv:2109.11405.
[18] Christopher M. Bishop, Pattern Recognition and Machine Learning, first ed., Springer-Verlag, New York, 2006.
[19] Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, second ed., Springer series in statistics, New York, 2009.
[20] Matthias M. Müller, Stefano Gherardini, Tommaso Calarco, Simone Montangero, Filippo Caruso, Information theoretical limits for quantum optimal control solutions: Error scaling of noisy channels, 2020, arXiv preprint arXiv:2006.16113.
[21] Akram Youssry, Gerardo A. Paz-Silva, Christopher Ferrie, Beyond quantum noise spectroscopy: modelling and mitigating noise with quantum feature engineering, Npj Quantum Inf. 6 (2020) 95.
[22] I.A. Luchnikov, S.V. Vintskevich, D.A. Grigoriev, S.N. Filippov, Machine learning non-Markovian quantum dynamics, Phys. Rev. Lett. 124 (14) (2020) 140502.
[23] Felipe F. Fanchini, Göktu ğ Karpat, Daniel Z. Rossatto, Ariel Norambuena, Raúl Coto, Estimating the degree of non-Markovianity using machine learning, Phys. Rev. A 103 (2021) 022425.
[24] Murphy Yuezhen Niu, Vadim Smelyanskyi, Paul Klimov, Sergio Boixo, Rami Barends, Julian Kelly, Yu Chen, Kunal Arya, Brian Burkett, Dave Bacon, et al., Learning non-Markovian quantum noise from moiré-enhanced swap spectroscopy with deep evolutionary algorithm, 2019, arXiv preprint arXiv:1912.04368.
[25] Robin Harper, Steven T. Flammia, Joel J. Wallman, Efficient learning of quantum noise, Nat. Phys. (2020) 1–5.
[26] Stefano Martina, Stefano Gherardini, Filippo Caruso, Machine learning approach for quantum non-Markovian noise classification, 2021, arXiv preprint arXiv:2101.03221.
[27] David F. Wise, John J.L. Morton, Siddharth Dhomkar, Using deep learning to understand and mitigate the qubit noise environment, PRX Quantum 2 (2021) 010316.
[28] Héctor Abraham, et al., Qiskit: An open-source framework for quantum computing, 2019.
[29] Michael A. Nielsen, Isaac Chuang, Quantum Computation and Quantum Information, American Association of Physics Teachers, 2002.
[30] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, Jay M Gambetta, Validating quantum computers using randomized model circuits, Phys. Rev. A 100 (3) (2019) 032328.
[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[32] Ivan H. Deutsch, Harnessing the power of the second quantum revolution, PRX Quantum 1 (2) (2020) 020101.
[33] K. Wright, K.M. Beck, S. Debnath, et al., Benchmarking an 11-qubit quantum computer, Nature Commun. 10 (2019) 5464.
[34] Jens Eisert, Dominik Hangleiter, Nathan Walk, Ingo Roth, Damian Markham, Rhea Parekh, Ulysse Chabaud, Elham Kashefi, Quantum certification and benchmarking, Nat. Rev. Phys. 2 (7) (2020) 382–390.
[35] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT press Cambridge, 2016, http://www.deeplearningbook.org.
[36] Jürgen Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117.