Good quantum kernels and where to find them

A. J. Ferreira–Martins[*] and C. Almeida[†]
Federal University of ABC

(Dated: October 24, 2021)

We propose the construction of a framework to systematically test and evaluate, for generic datasets, different feature map architectures yielding quantum kernel matrices, in a grid/random search-like approach. The framework is meant to be a pragmatic tool to assist research on Quantum Machine Learning.

Circumstantial evidence and some structural similarities between quantum computing and machine learning methods, as well as the extreme potential of these technologies, lead to a natural and very interesting question: what does quantum computing have to offer to machine learning? That is, is it possible to achieve quantum advantage for statistical learning tasks? These and other related questions build up the still young yet very active field of Quantum Machine Learning (QML).

One particular NISQ-friendly QML technique is that of variational/parameterized quantum circuits, leading to hybrid (classical-quantum) machine learning models [1]. If these models are applied to classification problems (which is the focus of this project), they are known as Variational Quantum Classifiers (VQC). These quantum circuits may be schematically thought of as composed of three layers:

- Feature map: first layer of the circuit, responsible for encoding classical data into quantum states which will be processed in the quantum circuit;
- Variational layer: parameterized part of the circuit. Parameters in this layer are learned in the training process;
- Measurement: final part of the circuit, consisting of measurements of the quantum register(s), thus producing classical information.

This construction comes with a very important caveat: the feature map and variational layers can be constructed in several different ways (that is, using different ansätze, consisting of different gates, in different configurations). A particular construction is also referred to as the "architecture" of the parameterized quantum circuit. Such immense freedom raises an important question: how could one propose an architecture which is suitable for a given problem? These questions are of major practical importance, and although some quite exciting results have been shown for very particular datasets [2, 3], no approach has yet been shown to yield good results which hold generically for arbitrary datasets.

The main goal of this project is to contribute directly to such pragmatic questions, by providing a framework capable of assisting the systematic search for good candidate architectures. However, we focus not in the full variational circuit, but rather in one of its components: the feature map!

Recent research has made clear that supervised quantum machine learning models are indeed kernel methods [2, 4]. This opens the way to very interesting questions concerning the theoretical properties of such quantum models, as well as practical questions. In this context, a very interesting possibility is the following: to use a quantum algorithm to compute a kernel (that is, building a kernel/Gram matrix), and then feed this pre-computed kernel matrix to a classical SVM algorithm to solve the quadratic problem.

Indeed, the feature map layer is named so because of what it implements: its goal is to encode classical data (from the input space) into the quantum circuit (a very particular and special feature space: the quantum Hilbert space). This opens the possibility to explore unique features of the quantum Hilbert space, namely, entanglement and superposition. Given that, one could argue that the resulting kernel (which we will refer to as quantum kernel) may have unique features which couldn't be achieved via classical methods. Indeed, that's one way in which we expect a quantum advantage: if there's a problem for which a "good" quantum kernel matrix can be constructed, and it is hard to simulate classically, it clearly yields a quantum advantage.

Now, the major aforementioned caveat is still present here: the feature map layer may also be constructed with a quite large freedom in what comes to its architecture — and this is the point which we plan to approach in this project!

The main question we want to answer is: how can we test different kernels, and tell if a given quantum kernel matrix is a good one?

From this, we propose the construction of a framework to systematically test different feature map architectures leading to quantum kernel matrices associated to a given dataset, in a grid/random search-like approach. We will use the Qiskit Machine Learning module to construct the feature maps and kernel matrices, and propose quantitative metrics to evaluate the quality of a given kernel. We then choose the best kernel, based on the quality metrics.

To our knowledge, the questions raised in this project

[*] andre.jfmdm@gmail.com
[†] caio.almeida.ads@gmail.com

still don't find any practical implementation to be answered. With the framework produced by this project, we hope to provide the community with such a practical tool (and even including it into Qiskit, if appropriate).

Therefore, the goal of this project is twofold: first, to provide a generic framework which allows one to systematically construct and check the quality of different quantum kernels proposals. Secondly, to offer a tool to easily do experiments which hopefully provide insights into what makes a good kernel good, leveraging the research on QML, more specifically to binary classification problems.

The framework consists of 5 main components. In what follows, we describe each one of these components in detail, each one in its respective section.

I. PRE-PROCESSING

We built this framework as a generic tool, to be applied to arbitrary datasets. However, given the nature of the operations performed, which includes a large number of simulations or executions of quantum circuits, it is very important that the dimensionality of the data that we work with is somehow adjusted. Otherwise, simulation or execution on NISQ-era hardware becomes impractical.

Thus, in order to guarantee generality, we included to the framework a procedure of subsampling, with the goal of providing the user with ways of reducing the dimensionality of the data prior to the actual quantum operations are performed — that is, as a pre-processing step.

When it comes to the construction of quantum kernels, it is very important to reduce as much as possible both the number of observations (training/testing examples) as well as the number of features/predictors. Indeed, the number of observations dictate how many quantum circuits must be executed in order to construct the kernel matrix. As simulating/executing these circuits may be costly, the smaller the number of observations, the better, as this implies a smaller number of circuit simulations/executions. On the other hand, the number of features determines the number of qubits making up each circuit. As the number of qubits increase, it becomes more costly or even impossible to simulate/execute the quantum circuits. Therefore, the smaller the number of features, the better, as this implies in a smaller number of qubits per circuit. This is why reductions in both dimensionalities is desirable.

Now, as it is always the case when such dimensionality reduction techniques are employed, it is important to do so whilst retaining most of the original information. With this in mind, we propose two options for this procedure:

- Subsampling from Determinantal Point Processes (DPPs): with DPPs we are able to construct subsamples of the original data, by selecting rows and/or columns from the data matrix in a repulsive manner, which means that it favors the sampling of rows/columns which are different among themselves, thus producing a representative, diverse subsample. We use the dppy python library [5]. Please refer to its documentation, or other relevant literature (for example [6]) for further details;

- Using Principal Components Analysis (PCA): with PCA, we are able to compute the so-called Principal Components, which are linear combinations of the original data matrix columns with a very particular property: they represent the directions which explain the data variance the most. Therefore, by choosing just a few of the first principal components, we hope to reduce the dimensionality of the features space quite dramatically, whilst retaining at least part of the original information, as all original features are present in the PC's linear combination. We use the implementation of scikit-learn [7] [8]. Notice that, in contrast to DPPs, we use PCA only as a way to reduce the number of features, not observations.

These pre-processing routines are presented as options for the user, so they're not mandatory. This means that, if one desires, it is in principle possible to run the framework for an arbitrary dataset with an arbitrary number of features and observations — all tools in the framework (except from some visualization functionalities) were made sufficiently generic as to accommodate any dimensionality. However, for computational costs reasons, dimensionality reduction is strongly advised. In the next section, we elaborate on the implications of the dimensionality of the data for the actual search procedures.

II. GRID/RANDOM SEARCH

The framework is meant to be a tool to systematically test different feature map architectures leading to quantum kernel matrices associated to a given dataset. To implement such a systematic approach, we provide 2 options: a grid search as well as a random search approach.

In the grid search approach, we consider and explicitly test all elements in the full space of parameters combinations. Conversely, in the random search approach we consider only a fixed number of combinations of the full space, thus explicitly picking a much smaller number of options to test. Grid search does access the full space of combinations, and because of that, it implies in much higher computational costs. Random search, on the other hand, is expected to yield good results whilst reducing dramatically the computational cost of the search.

If the grid search approach is chosen, we strongly advise against using a number of features larger than 4 and a number of observations larger than 100, as this would make the execution times of the framework way too long. However, if one desires to use dimensionalities larger than those, we strongly suggest the use of random search.

Now, in order to discuss the actual space of parameters to be tested, it is important to discuss how we shall construct the quantum feature maps.

III. QUANTUM FEATURE MAPS

For the contraction of quantum feature maps leading to quantum kernels, we shall use the `PauliFeatureMap` class [9] from Qiskit [10], and we will vary the following parameters:

- `eps`: the number of repeated blocks of the chosen feature map architecture;
- `entanglement`: specifies the entanglement structure associated with the feature map architecture;
- `paulis`: a list of strings for Pauli operators to be used in the feature map.

The general feature map to be implemented is given by the following operator [2]:

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} P_i \right) \quad , \tag{1}$$

where

$$\phi_S(\vec{x}) = \begin{cases} x_0 \text{ if } k = 1 \\ \prod_{j \in S}(\pi - x_j) \text{ otherwise} \end{cases} \quad . \tag{2}$$

And a particular feature map architecture is determined by a particular combination of the aforementioned parameters.

In order to exhaustively search for feature maps leading to good quantum kernels, we construct the full space of parameters allowed for `paulis` and `entanglement`. For `reps`, one could have, in principle, an infinite countable space (`reps` $\in \mathbb{N}$). However, we restrict this space to reps $\in \{1, 2, 3\}$, given that repetitions larger than 4 become increasingly difficult to simulate and/or execute on NISQ-era hardware.

Once the space of parameters for each one of the three aforementioned parameters is defined, we construct all possible combinations of them. However, some of these combinations, despite different, lead to identical feature maps. It is very important that such redundancies are eliminated, in order to reduce the space of parameters combinations as much as possible — after all, we perform an exhaustive search, so that if we can reduce the space of possibilities, we definitely should do so, to reduce computational costs. [11]

The actual quantum kernel matrix is recovered from the `QuantumKernel` class [12] from Qiskit, which takes as input the constructed feature map and a quantum instance or backend for simulation/execution. In the framework, we allow for both possibilities.

Up to now, we described how we will systematically construct quantum kernels. But, the framework is meant to actually determine, from all options considered in the search, which one yields the best result. In order to do so, we must have a way to actually evaluate the quality of a given quantum kernel. This is described in the next section.

IV. EVALUATION METRIC

In this project, we are interested in *binary classification* performed by classical SVM classifiers, using a quantum pre-computed kernel matrix. Given that, it is very reasonable that a kernel is a good one if it captures the following similarity structure: observations which belong to the same class are *similar* among themselves; and observations which are from different classes are *dissimilar* among themselves. In mathematical terms, consider two observations x_i and x_j. If they are from the same class ($y_i = y_j = 0$ or $y_i = y_j = 1$), we'd expect the respective Gram/kernel matrix component to be close to 1 $G_{ij} = \kappa_\Phi(x_i, x_j) \approx 1$; and if they are from different classes ($y_i \neq y_j$), we'd expect $G_{ij} = \kappa_\Phi(x_i, x_j) \approx 0$ (in this project, we consider a normalization such that $0 < G_{ij} < 1 \ , \forall i, j$).

Building on that, we propose an intuitive and visual way to assess the quality of kernel. First, order the observations so that observations of the same class are grouped together. This structures the Gram matrix in blocks:

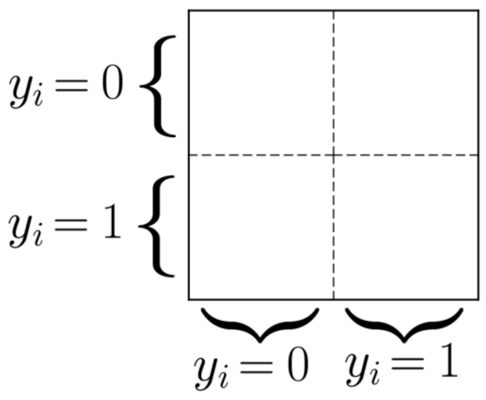

FIG. 1. Kernel matrix with observations grouped by their respective class.

Given that the training Gram matrix is symmetric ($G_{ij} = G_{ji}$), we can focus only on its lower diagonal:

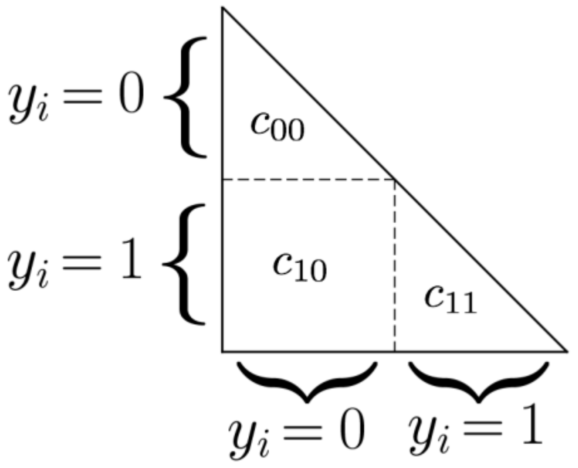

FIG. 2. Lower diagonal of a kernel matrix, which is symmetric.

Now, a "perfect" kernel matrix would have all of its components either 0 (in the bottom-left square region containing points of different classes) or 1 (in the remaining two triangles which contain points of the same class), that is,

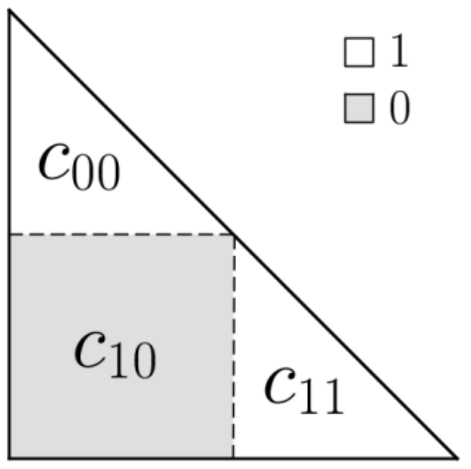

FIG. 3. Kernel matrix associated with a perfect kernel.

Real kernel matrices won't be that perfect, but, if a kernel matrix is a good one, we at least expect to be able to identify such a structure when it is plotted (like a heatmap, for instance). Such a visualization perfectly illustrates if the kernel is able to separate well the different classes or not. A "good" kernel matrix would look like this:

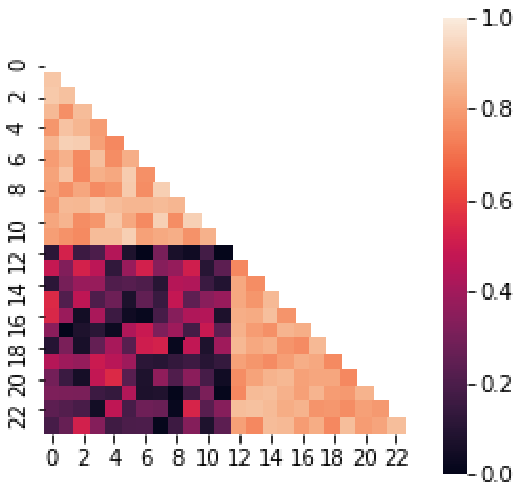

FIG. 4. Kernel matrix associated with a good kernel.

On the other hand, for a "bad" kernel matrix, it wouldn't be possible to identify the structure. It would look like this:

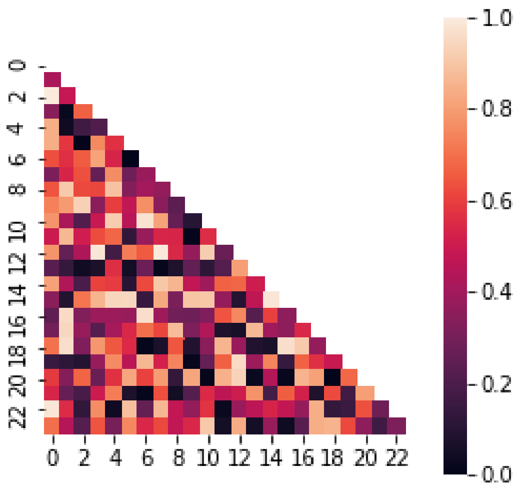

FIG. 5. Kernel matrix associated with a bad kernel.

Thus, by merely looking at plotted kernel matrices, we would be able to tell good ones from bad ones. That's a rather simple, intuitive and well-justified visual criterion. However, having to visually inspect possibly hundreds or thousands of plots really doesn't seem like a great idea. Thinking of that, we proposed a quantitative metric which aims at capturing precisely this same visual structure, but in a single number. Referring to the regions c_{00}, c_{11} and c_{10} as shown in Fig. 2, the proposed metric is

$$\Delta_\mu = \frac{1}{2}\left(\mu_{c_{00}} + \mu_{c_{11}}\right) - \mu_{c_{10}} \ , \tag{3}$$

where μ is simply the average of the elements in the respective region. We call each average a "density", which is why we refer to the metric Δ_μ as "densities difference".

Notice that for a perfect kernel (illustrated in Fig. 3), we'd have $\Delta_\mu = 1$, which is therefore the maximum value for Δ_μ. A quite bad kernel would be the opposite of the one in Fig. 3, such that $\mu_{c_{00}} = \mu_{c_{11}} = 0$ and $\mu_{c_{10}} = 1$ (that is, it considers observations of the same class quite dissimilar, whilst observations of different class quite similar). In this case, we'd have $\Delta_\mu = -1$. Given those extreme scenarios, it is clear that

$$-1 \leq \Delta_\mu \leq 1 \ . \tag{4}$$

This gives us a very forward way to valuate a kernel (and, most importantly, compare different kernels): the closer Δ_μ is to 1, the better the kernel; the closer Δ_μ is to -1, the worst. With this, we have established a meaningful metric to evaluate the kernels, and all we have to do is to calculate it for every (quantum) kernel matrix that is calculated in the grid/random search process, and then sort the kernels according to Δ_μ to pick the best ones.

Now, a very reasonable question is whether or not this proposed metric is indeed meaningful. That is, does it really captures the aforementioned visual behavior? Further, does it really capture the quality of a kernel itself?

In order to answer this question, we proposed a further step on the framework: we not only compute several kernels and their respective Δ_μ, but we also feed the resulting quantum kernel matrix as a pre-computed kernel to a classical SVM classifier estimator [13], train and evaluate it on test data, which yields several classification metrics (precision, recall and F1 score). With these classification metrics, we are then able to answer the following question: is Δ_μ correlated with the classification metrics? Indeed, it is reasonable that we expect that a "good kernel" must correlate strongly (and positively) with a good SVM model (which is, after all, the final goal). Thus, if such a strong positive correlation is observed between Δ_μ and the classification metrics, we could safely say that indeed it does capture the quality of a kernel.

As can be seen in the results notebooks [14], we found quite strong positive correlations between Δ_μ (`density_diff`) and the classification metrics. Therefore, we conclude that Δ_μ is indeed a good metric for the quality of a kernel.

Once the grid/random search executes through all the combinations, it produces a dataframe with all the aforementioned metrics (which is saved for later reference), as well as the respective feature map parameters. With this data, one is able to analyze several aspects of the results. We include such analysis as the final component of the framework.

V. ANALYTICS AND VISUALIZATION

As a final step, we produce some important analysis of the produced results, which serve as a way to check the validity of the proposed procedure as well as provide some insight into what makes a good kernel actually good; secondly, we also provide some interesting visualization tools.

First, we calculate and plot the correlation matrix of the results. In it, we have Δ_μ as well as the classification metrics and two other quantitative metrics which were proposed to quantify the feature map parameter `paulis`: the first one is `pauli_quantity`, which quantifies how many Pauli gates were considered in the parameter value; the second one is `pauli_diversity`, which quantifies how many different Pauli gates were considered. For example, for the parameter value `["X", "ZX"]`, we'd have `pauli_quantity` equals 3, and `pauli_diversity` equals 2. From our results, we found some really interesting points:

- Indeed, Δ_μ is a good kernel quality metric, as explained above;
- Δ_μ is almost uncorrelated to `reps`, the number of block repetitions on the feature map;
- Δ_μ is strongly and negatively correlated with `pauli_quantity`, indicating that fewer Pauli gates lead to better kernels (that's quite interesting!);
- Δ_μ also has a negative correlation with `pauli_diversity`, although not too strong. Also interesting!

Each result above is very interesting and could lead to lengthy discussions, which we won't do here. However, we hope this shows how much our framework does lead to some really interesting conclusions and possible insights on what makes good kernels good — and that's exactly one of our main goals with the project!

Finally, we also provide two interesting visualization tools: first, a way to visualize the decision boundary generated by the classical SVM trained with the quantum pre-computed kernel. This is very interesting because it allows one to study how the linear margin classifier on feature space projects onto the input space. Of course, this can only be done for 2-dimensional input spaces. The second visualization tool uses Qiskit's tools to visualize what happens with a given observation from the input space as it is mapped to Hilbert space by feature maps. The idea here is to visualize "the action" of the feature map (here we do what was schematically represented in the cover picture of the project). Although multi-qubit visualization is not the easiest task, we hope that this tool helps one understand what exactly a feature map does.

And that's all! This is the framework we propose. We hope all points were made clear. The full code is thoroughly commented for clarity of all major points, and is available in the online repository [15], along with all results. We sincerely hope that the framework becomes something useful for the QML research community. There are natural next steps for this project, and they will be addressed soon. In the mean time, if there is any question or suggestions, we're happy to receive them!

[1] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Parameterized quantum circuits as machine learning models, Quantum Science and Technology **4**, 043001 (2019), arXiv:1906.07682 [quant-ph].

[2] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Supervised learning with quantum-enhanced feature spaces, Nature (London) **567**, 209 (2019), arXiv:1804.11326 [quant-ph].

[3] Y. Liu, S. Arunachalam, and K. Temme, A rigorous and robust quantum speed-up in supervised machine learning, Nature Physics **17**, 1013 (2021), arXiv:2010.02174 [quant-ph].

[4] M. Schuld, Supervised quantum machine learning models are kernel methods, arXiv e-prints , arXiv:2101.11020 (2021), arXiv:2101.11020 [quant-ph].

[5] G. Gautier, G. Polito, R. Bardenet, and M. Valko, DPPy: DPP Sampling with Python, Journal of Machine Learning Research - Machine Learning Open Source Software (JMLR-MLOSS) (2019), code at http://github.com/guilgautier/DPPy/ Documentation at http://dppy.readthedocs.io/.

[6] A. Kulesza and B. Taskar, Determinantal point processes for machine learning, arXiv e-prints , arXiv:1207.6083 (2012), arXiv:1207.6083 [stat.ML].

[7] 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, and E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research **12**, 2825 (2011).

[8] Available here.

[9] Available here.

[10] M. S. ANIS, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, S. Anagolum, E. Arbel, A. Asfaw, A. Athalye, A. Avkhadiev, C. Azaustre, P. BHOLE, A. Banerjee, S. Banerjee, W. Bang, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, D. Bhatnagar, A. Bhobe, P. Bianchini, L. S. Bishop, C. Blank, S. Bolos, S. Bopardikar, S. Bosch, S. Brandhofer, Brandon, S. Bravyi, N. Bronn, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Carriker, I. Carvalho, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, F. Chevallier, K. Chinda, R. Cholarajan, J. M. Chow, S. Churchill, C. Claus, C. Clauss, C. Clothier, R. Cocking, R. Cocuzzo, J. Connor, F. Correa, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, N. D, S. Dague, T. E. Dandachi, A. N. Dangwal, J. Daniel, M. Daniels, M. Dartiailh, A. R. Davila, F. Debouni, A. Dekusar, A. Deshmukh, M. Deshpande, D. Ding, J. Doi, E. M. Dow, E. Drechsler, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, A. Ebrahimi, P. Eendebak, D. Egger, ElePT, Emilio, A. Espiricueta, M. Everitt, D. Facoetti, Farida, P. M. Fernández, S. Ferracin, D. Ferrari, A. H. Ferrera, R. Fouilland, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, T. Gates, L. Gil, A. Gilliam, A. Giridharan, J. Gomez-Mosquera, Gonzalo, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, H. Gupta, N. Gupta, J. M. Günther, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, A. Hasan, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, Ishwor, R. Iten, T. Itoko, A. Ivrii, A. Javadi, A. Javadi-Abhari, W. Javed, Q. Jianhua, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, J. Judge, T. Kachmann, A. Kale, N. Kanazawa, J. Kane, Kang-Bae, A. Kapila, A. Karazeev, P. Kassebaum, J. Kelso, S. Kelso, V. Khanderao, S. King, Y. Kobayashi, Kovi11Day, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, H. Landa, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, J. Lishman, D. Liu, P. Liu, Y. Maeng, S. Maheshkar, K. Majmudar, A. Malyshev, M. E. Mandouh, J. Manela, Manjula, J. Marecek, M. Marques, K. Marwaha, D. Maslov, P. Maszota, D. Mathews, A. Matsuo, F. Mazhandu, D. McClure, M. McElaney, C. McGarry, D. McKay, D. McPherson, S. Meesala, D. Meirom, C. Mendell, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, D. Miller, Z. Minev, A. Mitchell, N. Moll, A. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, D. Morcuende, S. Mostafa, M. Motta, R. Moyard, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, A. Ngoueya, J. Nicander, Nick-Singstock, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, T. Onodera, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, A. Panigrahi, V. R. Pascuzzi, S. Perriello, E. Peterson, A. Phan, F. Piro, M. Pistoia, C. Piveteau, J. Plewa, P. Pocreau, A. Pozas-Kerstjens, R. Pracht, M. Prokop, V. Prutyanov, S. Puri, D. Puzzuoli, J. Pérez, Quant02, Quintiii, I. R, R. I. Rahman, A. Raja, R. Rajeev, N. Ramagiri, A. Rao, R. Raymond, O. Reardon-Smith, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, Rietesh, D. Risinger, M. L. Rocca, D. M. Rodríguez, RohithKarur, B. Rosand, M. Rossmannek, M. Ryu, T. SAPV, N. R. C. Sa, A. Saha, A. Ash-Saki, S. Sanand, M. Sandberg, H. Sandesara,

R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, M. Schulterbrandt, J. Schwarm, J. Seaward, Sergi, I. F. Sertage, K. Setia, F. Shah, N. Shammah, R. Sharma, Y. Shi, J. Shoemaker, A. Silva, A. Simonetto, D. Singh, P. Singh, P. Singkanipa, Y. Siraichi, Siri, J. Sistos, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, V. P. Soloviev, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, A. Suau, S. Sun, K. J. Sung, M. Suwama, O. Słowik, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, K. Tian, M. Tillet, M. Tod, M. Tomasik, C. Tornow, E. de la Torre, J. L. S. Toural, K. Trabing, M. Treinish, D. Trenev, TrishaPe, F. Truger, G. Tsilimigkounakis, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. Vartak, A. C. Vazquez, P. Vijaywargiya, V. Villar, B. Vishnu, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, J. Wilson, E. Winston, WinterSoldier, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, M. Wright, L. Xing, B. Yang, D. Yeralin, R. Yonekura, D. Yonge-Mallo, R. Yoshida, R. Young, J. Yu, L. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hykavitha, itoko, jessica angel7, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, ryancocuzzo, saswati qiskit, septembrr, sethmerkel, shaashwat, sternparky, strickroman, tigerjack, tsura crisaldo, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and M. Čepulkovskis, Qiskit: An open-source framework for quantum computing (2021).

[11] Please refer to this notebook, in which we perform experiments and tests in order to argue how and in which cases it is possible to remove such redundancies.

[12] Available here.

[13] We use the implementation from sklearn.

[14] See this notebook for results of the framework applied to a particular dataset.

[15] All the code of the framework as well as some execution result are available in this online repository.