Quantum Kitchen Sinks
An algorithm for machine learning on near-term quantum computers

C. M. Wilson,[1, 2, 3] J. S. Otterbach,[1] N. Tezak,[1] R. S. Smith,[1] G. E. Crooks,[1] and M. P. da Silva[1]

[1]*Rigetti Computing, 2919 Seventh Street, Berkeley, CA, 94710-2704 USA*
[2]*Institute for Quantum Computing, University of Waterloo, Waterloo, N2L 3G1, Canada*
[3]*Department of Electrical and Computer Engineering,*
University of Waterloo, Waterloo, N2L 3G1, Canada
(Dated: June 22, 2018)

Noisy intermediate-scale quantum computing devices are an exciting platform for the exploration of the power of near-term quantum applications. Performing nontrivial tasks in such a framework requires a fundamentally different approach than what would be used on an error-corrected quantum computer. One such approach is to use *hybrid algorithms*, where problems are reduced to a parameterized quantum circuit that is often optimized in a classical feedback loop. Here we described one such hybrid algorithm for machine learning tasks by building upon the classical algorithm known as *random kitchen sinks*. Our technique, called *quantum kitchen sinks*, uses quantum circuits to nonlinearly transform classical inputs into features that can then be used in a number of machine learning algorithms. We demonstrate the power and flexibility of this proposal by using it to solve binary classification problems for synthetic datasets as well as handwritten digits from the MNIST database. We can show, in particular, that small quantum circuits provide significant performance lift over standard linear classical algorithms, reducing classification error rates from 50% to < 0.1%, and from 4.1% to 1.4% in these two examples, respectively.

Introduction— Interest in adapting or developing machine learning algorithms for near-term quantum computers has grown rapidly. While quantum machine learning (QML) algorithms offering exponential speed-ups on universal quantum computers have been known for some time [1–7], recent interest has increasingly focused on algorithms for noisy, intermediate-scale quantum (NISQ) computers [8–12]. These algorithms aim to minimize the complexity of the required quantum circuit so that they may be executed by NISQ devices while still yielding meaningful results. This is in contrast to approaches that allow for arbitrarily large circuits of width and depth that grow polynomially in the input size. These approaches can only yield meaningful answers if errors are suppressed to rates that are inversely proportional to the circuit size, something that is not possible with NISQ devices and requires fault tolerance [13–15].

Many of the proposed approaches use a so-called hybrid model for NISQ computing, where the quantum processor is considered an expensive resource and is extensively supported by classical computing resources. In particular, many of these proposals use a variational approach, where parameters of a small quantum circuit are optimized using classical optimization algorithms which use measurement outcomes to compute a cost function [11, 16–21]. While these closed-loop hybrid approaches move the computational cost of the optimization algorithm off of the quantum hardware, the iterative nature of the optimization process still requires a large number of calls to the "expensive" quantum resource.

In this paper, we propose a QML algorithm that eliminates the need for costly parameter optimization of quantum circuits. This novel open-loop hybrid algorithm, which we call *quantum kitchen sinks* (QKS), is inspired by a technique known as *random kitchen sinks* whereby random nonlinear transformation can greatly simplify the optimization of machine-learning (ML) tasks [22–24]. The general idea of QKS is to randomly sample from a family of quantum circuits and use each circuit to realize a nonlinear transformation of the input data to a measured bitstring. Subsequently, the concatenated results are processed with a classical machine learning (ML) algorithm. This approach is simple, flexible, and allows us to demonstrate that even small quantum circuits, deep in the NISQ regime, can provide significant "lift" for complex ML tasks such as the classification of hand-written digits. We further relate our circuits to common tools in ML known as kernels.

Random Kitchen Sinks— The objective in supervised ML is to approximate some *a priori* unknown function $f(\boldsymbol{u})$. For example, this function may be a map from images, represented by the variable $\boldsymbol{u}$, to labels, such as "cat" and "dog". This is often done by optimizing a parameterized function $g(\boldsymbol{u}; \boldsymbol{\theta})$ to maximize performance on a *training set* consisting of M examples $\{y_{i,\text{train}}, \mathbf{u}_{i,\text{train}}\}$ such that $g(\mathbf{u}_{i,\text{train}}; \boldsymbol{\theta}) \approx y_{i,\text{train}}$ for as many examples in the training set as possible. The quality of the approximation is often further quantified by how well g performs on some *test set* that is different from the training set [25]. A choice for g that perform particularly well is a deep neural network, which is a parametrized composition of many simple nonlinear functions, such as sigmoid functions or rectified linear units [26]. Finding the parameters of g that optimize performance (a process that for deep neural networks is known as *deep learning*) can be resource intensive, requiring large training sets and
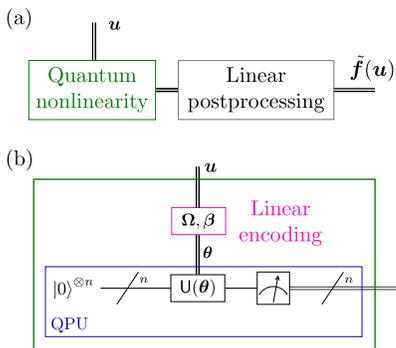
FIG. 1. (a) Quantum kitchen sinks approximate a function f applied to classical data **u** by using quantum circuits to apply a nonlinear transformation to **u** before additional classical (linear) postprocessing. (b) The classical data is transformed by first encoding into control parameters of a quantum circuit, and then measuring the quantum states. The results of measuring many different circuits parameterized by the same classical data is then collected into a single, large feature vector.

FIG. 2. QKS Ansätze for (a) two qubits using a CNOT, (b) two qubits using a CZ, and (c) four qubits. Circuit (a) and (b) are interesting to contrast because (a) leads to high performance classification in multiple datasets, while (b) leads to classifiers that are no better than random (see the text for an explanation). Larger circuits are described in the appendix.

and computational power [26].

Rahimi and Recht [22–24] observed that the costly optimization of the training process could be replaced by randomization. In an approach dubbed *random kitchen sinks* (RKS) [22–24], they showed it was possible to represent g as a weighted, linear sum of simple nonlinear functions that each have *random* parameters. Each term in this sum is called a "kitchen sink". The weights of the sum still need to be optimized, but this is a linear problem and, therefore, easy to solve. It has been shown that, for example, the cosine, sign (i.e., $d|x|/dx$), and indicator functions can be used to obtain good function approximations [22]. The RKS idea originated from an attempt to approximate the "kernel trick" [27, 28], by randomly sampling eigenfunctions of an integration kernel. Since then, this technique has been shown to apply beyond the sampling of a kernel, and to deliver performance that is comparable to deep learning, while relying on much simpler numerical techniques [29, 30].

The performance of the algorithm is dependent on the number of kitchen sinks D, the choice of nonlinear function, as well as the number of training examples M. Rahimi and Recht showed the the approximation error of g in RKS scales as $O(\frac{1}{\sqrt{D}} + \frac{1}{\sqrt{M}})$ [24] such that it may be necessary to have large training sets and to generate many RKS in order to achieve the same error rate as standard kernel methods [31].

Classical vs. quantum power— Before discussing how to generalize RKS to a quantum setting, we would like to make an important observation. In proposing a ML algorithm for quantum computers, there is a danger that the quantum processor does not contribute in a meaningful way to the power of the technique. If the external c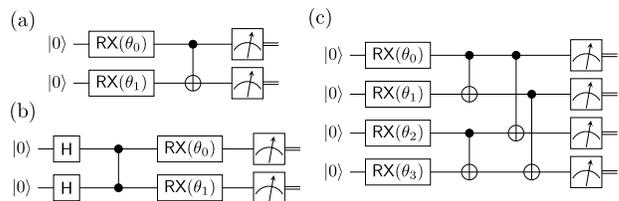lassical part is powerful enough, the algorithm may work *in spite of* the transformation made by the quantum processor. This can be seen as the flip-side of the RKS result we adapt: generic nonlinearities in the *classical* processing can add power to the ML algorithm, even if the quantum processing does not. For this reason, it is important in a research context that the classical portion of the algorithm be as simple and linear as possible. For this reason, we will require all classical pre- and postprocessing to be strictly linear, and consider only the added power of a nonlinear transformation enabled by the quantum processor. We will refer to this as the *Linear Baseline (LB) Rule*.

Applying the LB Rule to our strategy for testing and validating our algorithm, we design an algorithms such that the quantum processor can be removed and the input data can be passed directly through the remaining (linear) classical part of the algorithm. We can then benchmark the performance lift provided by the quantum processor against the performance of the classical algorithm on its own. Note that a lift provided by the quantum processor in this context does not imply an absolute quantum advantage, but it does gives us a simple, operational method to identify the power added by the quantum circuit.

Quantum Kitchen Sinks— We now describe our approach to translate the RKS framework into something that may be computed by a quantum computer—what we call QKS (see Fig. 1).

As noted above, one of the nonlinear functions used to build RKSs is a cosine. We can easily generate cosine transformations in a quantum setting by applying a Rabi rotation to a single qubit with a rate and phase that is chosen at random, but a time duration that is a function of the input data. While this quantum construction of RKSs works well, it can easily be simulated on a classical computer.

In order to generalize this to circuits that are harder to simulate, our first step is to specify the input data encoding in more detail. We choose to encode the data into angles of rotations in the quantum circuit, while keeping the state preparation and measurement fixed (a similar

approach was taken in [11] for a different QML technique). This naturally leads the measurement statistics to depend nonlinearly on the classical data.

Under the LB rule, we require that the mapping from data to angles be linear. To define a linear encoding, let $\mathbf{u}_i \in \mathbb{R}^p$ for $i = 1, \ldots, M$ be a p-dimensional input vector from a data set containing M examples. We can encode this input vector into q gate parameters using a $(q \times p)$-dimensional matrix $\mathbf{\Omega}_e$ of the form $\mathbf{\Omega}_e = (\omega_1, \ldots, \omega_q)^\mathsf{T}$ where ω_k is a p-dimensional vector with a number $r \leq p$ elements being random values and the other elements being exactly zero. We can also specify a random q-dimensional bias vector $\boldsymbol{\beta}_e$. We then get our set of random parameters $\boldsymbol{\theta}_{i,e}$ from the linear transformation $\boldsymbol{\theta}_{i,e} = \mathbf{\Omega}_e \mathbf{u}_i + \boldsymbol{\beta}_e$. Notice the additional index e which denotes the eth episode, i.e., the eth repetition of the circuit parameterized through the encoding $\mathbf{\Omega}_e, \boldsymbol{\beta}_e$ (see below for a discussion about episodes).

By specifying different elements of ω_k to be nonzero, we can specify different encodings. For instance, we can encode a p-dimensional input vector into a single-qubit circuit by choosing $q = 1$ and $r = p$. In this single-qubit encoding, all dimensions of $\mathbf{u}_i$ are combined into a single control parameter. Conversely, we could use a split encoding with $q = p$ and $r = 1$, where each dimension of $\mathbf{u}_i$ is fed into a distinct control parameter. We discuss other possibilities below. Note that the set of encoding parameters $\{\mathbf{\Omega}_e, \boldsymbol{\beta}_e\}_{e=1}^{E}$ is only drawn once and becomes a static part of the machine, which is used for both training and testing. For the results presented in this paper, the nonzero elements of $\mathbf{\Omega}_e$ are drawn from a zero-mean normal distribution with variance σ^2, i.e., $\mathcal{N}(0, \sigma^2)$ and the elements of $\boldsymbol{\beta}_e$ are drawn from a uniform distribution $\mathcal{U}(0, 2\pi)$ [32]. However, other distributions may also be considered. These choices only partially determine the encoding. The exact structure of the circuit and how the parameters ω_k parameterizes the circuit will also have an impact on the performance of the algorithm, and illustrate the large flexibility available for designing QKSs tailored to particular datasets and applications.

The choice of distributions and the parameterization of the circuit together implicitly define a kernel which allows for QKSs to be analyzed as a standard kernel machine, as we describe later. The computation of the kernel is not necessary for the use of the QKS, and in fact may require exponentially large resources, but it may be helpful in designing the circuit Ansätze.

Once we have encoded the data into control parameters, we are ready to preprocess the data. Since the input data is encoded in circuit parameters, the choice of input state is somewhat arbitrary. For simplicity and without loss of generality, we choose the all-zeros state $|\Psi_{\text{in}}\rangle = |00 \ldots\rangle$. Since any other input state would be generated by another quantum circuit, the composition of this circuit with the QKS encoding would correspond to a different circuit Ansatz.

In order to postprocess the QKS output, we must also extract classical data from the state. This is done by simply measuring the state in the computational basis—again, without loss of generality, since a basis transformation would simply translate into changing the circuit Ansatz. The output of the measurements gives us classical bits. We have some design freedom in choosing how to (classically) process these output bits into features. Under the LB rule, care should be taken in this choice such that nonlinear postprocessing is avoided. For this work, we will simply "stack" all of the bits into a q-dimensional feature vector.

Contrary to the RKS approach, this feature mapping is stochastic. Our proposal does not preclude averaging over many shots of the same circuit, but the numerical studies described here use only individual shots of each circuit.

Once we have constructed our feature vectors, they are fed into a classical machine learning algorithm, which under the LB rule, we take to be linear (as is also the case in RKS).

It is well-known in machine learning that transforming data into a higher-dimensional feature space can be useful. There are two strategies to generate higher-dimensional features using QKS: entangling more and more qubits, or generating more and more random circuits. The first strategy leads straightforwardly to a quantum advantage argument if the parameterized circuits used are hard to simulate [33–36]. However, large, monolithic circuits may also require very low error rates. The second strategy is more readily scaled in NISQ devices, and it simply requires running E fixed circuits, which we call *episodes*, to obtain a feature vector that is $(E \times q)$-dimensional for q qubits. We expect D parameters in RKS should be roughly equivalent to $E \times q$ parameters in QKS, but we do not have formal results that guarantee this correspondence.

A synthetic example— As an example to demonstrate the effectiveness of QKS, we incorporate it into a standard binary classification problem. As our classical, linear baseline, we use the logistic regression (LR) classifier provided by the `scikit-learn` package. As a first data set, we chose the synthetic "picture frames" data set shown in Fig. 3. The data set was chosen to have two classes that are not separable by a linear boundary. The training set contained $M = 1600$ two-dimensional points, 800 for each class. The classification accuracy was tested using a different set of 400 points arranged in a similar configuration.

We coded the algorithm using the pyQuil® Python package [37, 38] and executed it on the Rigetti QVM™, available through the Forest platform [39]. The QVM is a high-performance quantum simulator written in ANSI Common Lisp [40]. In order to run the numerical experiments in conjunction with post-processing software in Python, the QVM was extended with a new entry-
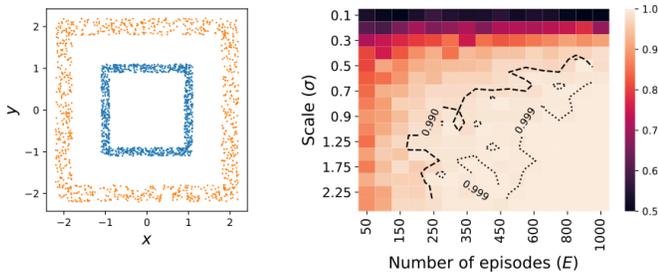
FIG. 3. (a) Synthetic "picture frames" dataset. (b) Performance of the QKS classifier combined with logistic regression. We show the result of optimizing the performance as a function of the hyperparameters σ and E. The contours separate orders of magnitudes in error rate. We see that optimal performance (with a test accuracy of $> 99.9\%$) is achieve with $\sigma \approx 1$.

point to allow high-speed execution of a large number of episodes (on the order of 10^4) for a given circuit Ansatz and input $\mathbf{u}$. In particular, the QVM was extended so that a template Quil [38] program defined with the DEFCIRCUIT facility could be supplied along with a collection of DEFCIRCUIT parameter tuples. The QVM reads these parameter tuples, fills them into the supplied program in constant time, and executes the resulting program, all while eliminating unnecessary memory accesses and allocations. This modification to the QVM was made possible using Quil's hybrid classical/quantum memory model. See the Appendix (e.g., Fig. 8) for examples of circuit Ansätze written in Quil.

Applying the baseline LR algorithm to the picture frame dataset yielded a classification accuracy of approximately 50%, meaning it performs no better than randomly assigning classes to each point. We then used the QKS construction, using the circuit shown in Fig. 2. For the data presented here, we used split encoding (defined above) with $q = p = 2$ and $r = 1$, and optimized over the number of episodes E and the parameter σ used in the random encoding. The best classification accuracy achieved was $> 99.9\%$, a remarkable performance lift over the linear baseline, illustrating the power of QKS (see Fig. 3).

A real-world example— While this synthetic example illustrates the computational power provided by the QKS, it is interesting to consider a less structured classification problem originating in the real world: discriminating hand-written digits from the MNIST dataset [41]. This dataset is a well-known benchmark in machine learning. While it is a multiclass problem, we choose to focus on classifying two digits that are difficult to distinguishing using LR: "3" and "5". The classification accuracy we obtain with LR is 95.9%, which will serve as our linear baseline.

The MNIST dataset has a much higher dimensionality than the previous example. Each digit is a (28×28)-pixel 8-bit grayscale image, so care must be taken to encode
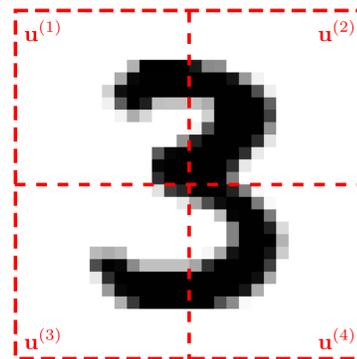

FIG. 4. A four-qubit partitioning of an example MNIST digit. Each partition, called a *tile*, corresponds to disjoint collection of components of the input vector $\mathbf{u}$, i.e., $\mathbf{u}$ is some permutation of the vector $(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}, \mathbf{u}^{(4)})^{\mathsf{T}} \in \mathbb{R}^{784}$. The exact permutation is encoded in the choice of nonvanishing values of the matrix $\boldsymbol{\Omega}$.

the data into a small number of qubits. A standard first step is to vectorize the image, by stacking the columns of the image into a $p = 784$ dimensional vector. We use a slightly modified approach intended to preserve more of the spatial structure of the image. After standardizing the image [42], to run MNIST on a q-qubit processor, we first split each image into q rectangular tiles, and construct fixed-depth circuit Ansätze where only single-qubit gates have parameterized rotations (see Fig. 4). The encoding vectors ω_k are then chosen to have blocks of $r = p/q$ nonzero elements that select out values of only one tile per gate parameter.

With this encoding, we have simulated the performance of QKS on the $(3, 5)$-MNIST dataset for different numbers of qubits. The best error rate is 1.4%, which is a reduction of the error rate by more than a factor of 2 compared to the linear baseline.

In Fig. 5 we plot the minimum error rate for classifying the $(3, 5)$-MNIST dataset using QKS for different numbers of qubits [43]. Comparing the results for different number of qubits, there is a clear minimum in the error rate in the range 2 to 4 qubits. For more qubits, the error rate increases again. There are a number of possible explanations for this behavior. One possibility is simply that the particular circuits chosen (including the CNOT networks) may be suboptimal for this task. Another possibility is that the MNIST data set is insufficiently large to properly train the larger number of parameters in the larger circuits, leading to overfitting. These and other hypotheses will be explored in future work.

It is also important to point out that quantum coherence does not play a role in "single layer" circuits, as the same output mapping can be implemented using purely classical stochastic processes (i.e., the circuits are efficiently simulatable, in the weak sense [44]). We are able to show, however, that by increasing the number of lay-
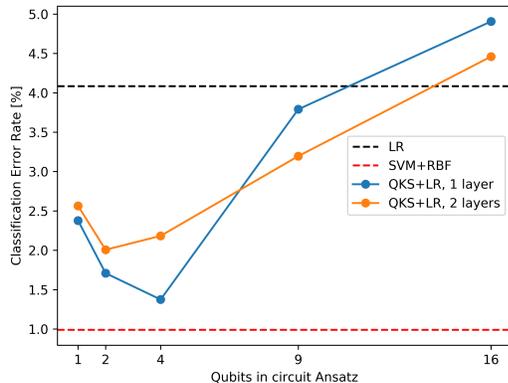
FIG. 5. Scaling of the error rate classifying the $(3,5)$-MNIST dataset using QKS combined with logistic regression (LR), as a function of the numbers of qubits. As a reference, we include the performance of LR on its own (our linear baseline) and the performance of a nonlinear classifier built out of a support vector machine (SVM) with a radial basis function (RBF) kernel. Details of the circuit Ansätze can be found in Fig. 2 and the appendix.

ers of the circuit Ansätze (each layer using independently chosen linear encodings) we are able to maintain similar performance (see Fig. 5). One may also consider other Ansätze based on circuits that are conjectured to be hard to simulate [33–36].

The implied kernels— The random sampling of non-linear feature maps across different episodes can be connected to the use of an implicit kernel function [22–24]. Formally, the kernel is the inner product between input vectors after their nonlinear mapping by the kitchen sinks. Informally, the kernel function $k(\mathbf{u}, \mathbf{v})$ of two input vectors $\mathbf{u}$ and $\mathbf{v}$ expresses the similarity between these inputs. Even though the random and quantum kitchen sinks do not explicitly use the kernel, it is instructive to calculate the implicit kernel associated with our circuits, as it can point to better ways to build circuit Ansätze, and consider the effect of noise.

We compute the implicit kernel by evaluating the inner product of two binary feature vectors sampled using a QKS circuit. Let $\mathbf{b}_{e,\mathbf{u}}(\boldsymbol{\theta}_e)$ and $\mathbf{b}_{e,\mathbf{v}}(\boldsymbol{\theta}_e)$ denote the vectorized output of a single episode e with the random parameters $\boldsymbol{\theta}_e$ on the inputs $\mathbf{u}$ and $\mathbf{v}$. The inner product of the total feature vector can then be computed as

$$\tilde{k}(\mathbf{u}, \mathbf{v}) = \frac{1}{E} \sum_{e=1}^{E} \mathbf{b}_{e,\mathbf{u}}(\boldsymbol{\theta}_e) \cdot \mathbf{b}_{e,\mathbf{v}}(\boldsymbol{\theta}_e)$$

where we have added the normalization by E.

The quantities $\mathbf{b}_{e,\mathbf{v}}(\theta_e)$ are random variables with bit-string values $z \in \{0,1\}^q$. The probability of a given outcome z is $p_{e,\mathbf{u}}^{(z)} = |\langle z| U(\mathbf{u}, \theta_e) |\Psi_{\text{in}}\rangle|^2$ where $U(\mathbf{u}, \theta_e)$ is the unitary transformation realized by the QKS circuit.

We then find

$$\langle \mathbf{b}_{e,\mathbf{u}} \cdot \mathbf{b}_{e,\mathbf{v}} \rangle = \sum_{s=0}^{q} s P(\mathbf{b}_{e,\mathbf{u}} \cdot \mathbf{b}_{e,\mathbf{v}} = s) = \mathbf{p}_{e,\mathbf{u}}^{\mathsf{T}} \mathbf{S} \, \mathbf{p}_{e,\mathbf{v}}$$

where the matrix $\mathbf{S}$ contains the inner product of the bit strings z and z', and the vector $\mathbf{p}_{e,\mathbf{u}}$ the outcome probabilities, both indexed by z.

We now note that since the parameters $\boldsymbol{\theta}_e$ are drawn from a classical probability distribution $P(\boldsymbol{\theta})$, we can view the sum $\tilde{k}(u, v)$ as a Monte Carlo estimator. In the limit of an infinite number of episodes ($E \rightarrow \infty$), the kernel then approaches the form

$$k(\mathbf{u}, \mathbf{v}) = \int \mathrm{d}\boldsymbol{\theta} \; P(\boldsymbol{\theta}) \; \mathbf{p}_{\mathbf{u}}^{\mathsf{T}}(\boldsymbol{\theta}) \, \mathbf{S} \, \mathbf{p}_{\mathbf{v}}(\boldsymbol{\theta}). \quad (1)$$

Using this result, we can, for instance, calculate the implicit kernel for the circuit in Fig. 2(a). To do so, we specify that the values of the matrix Ω are drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$ and that the elements of the bias vector $\boldsymbol{\beta}_e$ are drawn from a uniform distribution $\mathcal{U}(0, 2\pi)$. Using (1), we then find the implicit kernel as

$$k(\mathbf{u}, \mathbf{v}) = \frac{1}{2} + \frac{1}{8}e^{-\frac{1}{2}\sigma^2\|\mathbf{u}^{(1)} - \mathbf{v}^{(1)}\|_2^2} + \frac{1}{16}e^{-\frac{1}{2}\sigma^2\|\mathbf{u} - \mathbf{v}\|_2^2}, \quad (2)$$

where $\mathbf{u}^{(i)}$ ($\mathbf{v}^{(i)}$) is the ith tile (out of 2) of the input data vector $\mathbf{u}$ ($\mathbf{v}$) [45]. We see that the last term here is a radial basis function (RBF) kernel that is standard in machine learning. There are additional components, including a constant term. The second term depends only on part of the data. Similar calculations can be performed for the other circuit Ansätze, and again we find multiple terms that depend on different subsets of the data. One can imagine optimizing the CNOT network to maximize sensitivity to the most relevant subsets of the data, but we do not explore the possibility here.

Interestingly enough, not all circuit Ansätze lead to a useful kernel. For instance, circuit Fig. 2(b) seems similar to the just-analyzed circuit. However, if we calculate the implied kernel of this circuit, we find the constant function $k(\mathbf{u}, \mathbf{v}) = 1/2$, independent of the input vectors $\mathbf{u}$ and $\mathbf{v}$. This suggests that this circuit should have no discrimination power and, in fact, our numerical results confirm this.

We then see that QKS provides a rich structure to construct implicit kernels, with not only the choice of circuit, but the choice of encoding, choice of decoding, and choice of probability distributions shaping the kernel in understandable ways.

Conclusion— We have described how random quantum circuits can be used to transform classical data in a highly nonlinear yet flexible manner, similar to the random kitchen sinks technique from classical machine learning. These transformations, which we dub *quantum kitchen sinks*, can be used to enhance classical machine learning algorithms. We illustrated this enhance-

ment by showing that the accuracy of a logistic regression classifier can be boosted from 50% to $> 99.9\%$ in low-dimensional synthetic datasets, and from 95.9% to 98.6% in a high-dimensional dataset consisting of the hand-written "3" and "5" digits of the MNIST database. In all these examples, this can be achieved with as few as four qubits. Future work will focus on exploring different circuit Ansätze, and developing a better understanding of the performance of this technique.

Contributions— CMW first conceived of QKS. CMW and JO developed the theory and prototyped the numerical analysis. JO, NT, and RSS developed the scalable analysis for larger datasets. GEC proposed the LB rule. MPS supervised and coordinated the effort. CMW, JO, NT, RSS, and MPS wrote the manuscript.

[1] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).

[2] P. Rebentrost, M. Mohseni, and S. Lloyd, Phys. Rev. Lett. **113**, 130503 (2014).

[3] S. Lloyd, M. Mohseni, and P. Rebentrost, Nature Physics **10**, 631 EP (2014).

[4] A. W. Cross, G. Smith, and J. A. Smolin, Phys. Rev. A **92**, 012327 (2015).

[5] I. Kerenidis and A. Prakash, "Quantum recommendation systems," (2016), arXiv:1603.08675.

[6] F. G. S. L. Brandão and K. M. Svore, in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)* (2017) pp. 415–426.

[7] F. G. S. L. Brandão, A. Kalev, T. Li, C. Y.-Y. Lin, K. M. Svore, and X. Wu, "Quantum SDP solvers: Large speedups, optimality, and applications to quantum learning," (2017), arXiv:1710.02581.

[8] J. Preskill, "Quantum computing in the NISQ era and beyond," (2018), arXiv:1801.00862.

[9] M. Schuld and et al, (2018), arXiv:1804.00633.

[10] E. Grant and et al., (2018), arXiv:1804.03680.

[11] V. Havlicek and et al., (2018), arXiv:1804.11326.

[12] W. Huggins and et al., (2018), arXiv:1803.11537.

[13] E. Knill, R. Laflamme, and W. H. Zurek, Science **279**, 342 (1998).

[14] P. Aliferis, D. Gottesman, and J. Preskill, Quantum Info. Comput. **6**, 97 (2006).

[15] D. Aharonov and M. Ben-Or, SIAM J. Comput. **38**, 1207 (2008).

[16] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, Nature Communications **5**, 4213 EP (2014).

[17] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," (2014), arXiv:1411.4028.

[18] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, Nature **549**, 242 EP (2017).

[19] E. Farhi, J. Goldstone, S. Gutmann, and H. Neven, "Quantum algorithms for fixed qubit architectures," (2017), arXiv:1703.06199.

[20] Z.-C. Yang, A. Rahmani, A. Shabani, H. Neven, and C. Chamon, Phys. Rev. X **7**, 021027 (2017).

[21] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," (2018), arXiv:1802.06002.

[22] A. Rahimi and B. Recht, in *2008 46th Annual Allerton Conference on Communication, Control, and Computing* (2008) pp. 555–561.

[23] A. Rahimi and B. Recht, in *Advances in Neural Information Processing Systems 20*, edited by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Curran Associates, Inc., 2008) pp. 1177–1184.

[24] A. Rahimi and B. Recht, in *Advances in Neural Information Processing Systems 21*, edited by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Curran Associates, Inc., 2009) pp. 1313–1320.

[25] The use of a test set helps to avoid so-called "overfitting" of g to the training set, as one would like an approximation to f that generalizes well to previously unseen data, not simply an approximation that works only on the training set.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) http://www.deeplearningbook.org.

[27] B. Schölkopf and A. Smola, *Learning with kernels : support vector machines, regularization, optimization, and beyond* (MIT Press, Cambridge, Mass, 2002).

[28] T. Hofmann, B. Schölkopf, and A. J. Smola, Ann. Statist. **36**, 1171 (2008).

[29] A. May, A. Bagheri Garakani, Z. Lu, D. Guo, K. Liu, A. Bellet, L. Fan, M. Collins, D. Hsu, B. Kingsbury, M. Picheny, and F. Sha, (2017), arXiv:1701.03577.

[30] "Reflections on random kitchen sinks," Accessed: 2018-06-01.

[31] This means, in particular, that if the kernel corresponding to a circuit Ansatz is known, it is possible to estimate the RKS error rate for that Ansatz by using a classical kernel machine, although performance as a function of D is often better than what would be expected from these bounds [30].

[32] Note that the hyperparameter σ must be optimized during training.

[33] S. Aaronson and A. Arkhipov, in *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11 (ACM, New York, NY, USA, 2011) pp. 333–342.

[34] E. Farhi and A. W. Harrow, "Quantum supremacy through the quantum approximate optimization algorithm," (2016), arXiv:1602.07674.

[35] M. J. Bremner, A. Montanaro, and D. J. Shepherd, Phys. Rev. Lett. **117**, 080501 (2016).

[36] M. J. Bremner, A. Montanaro, and D. J. Shepherd, Quantum **1**, 8 (2017).

[37] Rigetti Computing, "pyQuil," https://github.com/rigetticomputing/pyquil (2016).

[38] R. S. Smith, M. J. Curtis, and W. J. Zeng, (2016), arXiv:1608.03355.

[39] Rigetti Computing, "Forest," https://www.rigetti.com/forest (2017).

[40] American National Standards Institute and Information Technology Industry Council, *American National Standard for Information Technology: programming language — Common LISP*, American National Standards Insti-

tute, 1430 Broadway, New York, NY 10018, USA (1996), approved December 8, 1994.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proceedings of the IEEE **86**, 2278 (1998).

[42] *Standardization* or *z-normalization* of a set $X \subset \mathbb{R}$ is the pointwise map $x \mapsto (x - \mathrm{E}[X])/\mathrm{Var}(X)^2$.

[43] Each point corresponds to the best accuracy observed for a range of σ and number of episodes, much like what is shown in Fig. 3. This procedure is known as "hyperparameter optimization", and is a common step in ML algorithms.

[44] M. Van Den Nest, Quantum Info. Comput. **10**, 258 (2010).

[45] For the picture frame data set, this is just the ith vector component.

Picture Frames Dataset

The picture frames dataset (Fig. 3) was chosen to have a nontrivial shape and such that the two classes were not linearly separable. The smaller (red) square has a side length of 2 with points uniformly distributed in a region 0.1 around the average. The larger (blue) square has a side length of 4 with points uniformly distributed in a region 0.2 around the average.

Parameterized programs for circuits Ansätze

Figs. 6, 7, and 8 define circuit Ansätze for 4, 8, and 16 qubits respectively using the DEFCIRCUIT facility in Quil [38]. DEFCIRCUIT defines a template which can be filled in via the %-prefixed parameters.

```
DEFCIRCUIT P4(%x0,%x1,%x2,%x3):
    RX(%x0) 0
    RX(%x1) 1
    RX(%x2) 2
    RX(%x3) 3
    CNOT 0 2
    CNOT 1 3
    CNOT 0 1
    CNOT 2 3
```

FIG. 6. A four-qubit QKS Ansatz written using a Quil DEFCIRCUIT.

```
DEFCIRCUIT P9(%x0,%x1,%x2,%x3,%x4,%x5,%x6,%x7,%x8):
    RX(%x0) 0
    RX(%x1) 1
    RX(%x2) 2
    RX(%x3) 3
    RX(%x4) 4
    RX(%x5) 5
    RX(%x6) 6
    RX(%x7) 7
    RX(%x8) 8
    CNOT 0 3
    CNOT 1 4
    CNOT 2 5
    CNOT 3 6
    CNOT 0 1
    CNOT 3 4
    CNOT 5 8
    CNOT 6 7
    CNOT 1 2
    CNOT 4 7
    CNOT 4 5
    CNOT 7 8
```

FIG. 7. A nine-qubit QKS Ansatz written in Quil.

```
DEFCIRCUIT P16(%x0,%x1, ..., %x14,%x15):  # params elided
    RX(%x0) 0
    RX(%x1) 1
    RX(%x2) 2
    RX(%x3) 3
    RX(%x4) 4
    RX(%x5) 5
    RX(%x6) 6
    RX(%x7) 7
    RX(%x8) 8
    RX(%x9) 9
    RX(%x10) 10
    RX(%x11) 11
    RX(%x12) 12
    RX(%x13) 13
    RX(%x14) 14
    RX(%x15) 15
    CNOT 0 4
    CNOT 1 5
    CNOT 2 6
    CNOT 3 7
    CNOT 8 12
    CNOT 9 13
    CNOT 10 14
    CNOT 11 15
    CNOT 0 1
    CNOT 2 3
    CNOT 4 5
    CNOT 6 7
    CNOT 8 9
    CNOT 10 11
    CNOT 12 13
    CNOT 14 15
    CNOT 1 2
    CNOT 4 8
    CNOT 5 9
    CNOT 6 10
    CNOT 5 6
    CNOT 7 11
    CNOT 9 10
    CNOT 13 14
```

FIG. 8. A 16-qubit QKS Ansatz written in Quil.