Training Quantum Embedding Kernels on Near-Term Quantum Computers

Thomas Hubregtsen,[1,*] David Wierichs,[2] Elies Gil-Fuster,[1]
Peter-Jan H. S. Derks,[1] Paul K. Faehrmann,[1] and Johannes Jakob Meyer[1,3]

[1]*Dahlem Center for Complex Quantum Systems, Freie Universität Berlin, 14195 Berlin, Germany*
[2]*Institute for Theoretical Physics, University of Cologne, 50937 Cologne, Germany*
[3]*QMATH, Department of Mathematical Sciences, University of Copenhagen, 2100 Copenhagen, Denmark*

(Dated: May 7, 2021)

Kernel methods are a cornerstone of classical machine learning. The idea of using quantum computers to compute kernels has recently attracted attention. *Quantum embedding kernels (QEKs)* constructed by embedding data into the Hilbert space of a quantum computer are a particular quantum kernel technique that allows to gather insights into learning problems and that are particularly suitable for noisy intermediate-scale quantum devices. In this work, we first provide an accessible introduction to quantum embedding kernels and then analyze the practical issues arising when realizing them on a noisy near-term quantum computer. We focus on quantum embedding kernels with variational parameters. These variational parameters are optimized for a given dataset by increasing the kernel-target alignment, a heuristic connected to the achievable classification accuracy. We further show under which conditions noise from device imperfections influences the predicted kernel and provide a strategy to mitigate these detrimental effects which is tailored to quantum embedding kernels. We also address the influence of finite sampling and derive bounds that put guarantees on the quality of the kernel matrix. We illustrate our findings by numerical experiments and tests on actual hardware.

I. INTRODUCTION

Quantum computing promises to solve problems that are currently intractable by exploiting the quantum nature of information. While long considered more a dream than a possibility, recent efforts have succeeded in constructing quantum devices able to perform computations intractable for classical computers [1]. This generation of quantum devices is referred to as *noisy intermediate-scale quantum* (NISQ) devices. Exploiting the non-classical capabilities of NISQ devices to solve practically relevant problems is a rapidly growing field of research [2, 3].

Machine learning (ML) on the other hand promises to leverage classical computers to solve ever more complicated problems. Especially the combination of deep neural networks and big data has led to impressive successes recently [4–6].

There exists, however, a variety of different approaches to construct learning models that are currently outshone by the more popular deep neural networks. Among those, *kernel methods* are of particular interest as they provide a way to realize machine learning models that come with strong guarantees on their performance and offer a deep theoretical understanding that is often lacking when dealing with deep neural networks [7].

It is no surprise that the idea of using near-term quantum computers for machine learning – dubbed *quantum machine learning* (QML) – has gained considerable traction lately [8–10]. The most prominent approach to construct learning models using NISQ devices relies on the use of *parametrized quantum circuit*s (PQCs) [11–14]. Kernel methods in particular have emerged as one particular candidate to realize QML models [15–21]. Furthermore, it was recently shown that other types of variational quantum learning models are fundamentally related to quantum kernel methods [22] and that quantum kernels enable the construction of learning problems that prove a separation between classical and quantum machine learning [20, 23]. This work focuses specifically on *quantum embedding kernel*s (QEKs), a subclass of quantum kernel methods where a PQC is used to embed datapoints into the Hilbert space of quantum states of the underlying NISQ device.

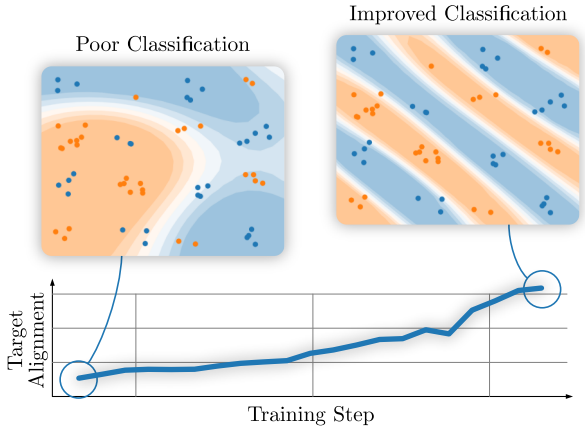

Figure 1. Quantum embedding kernels allow for classification of datasets through the use of a support vector machine. The quality of classification can be significantly improved by optimizing the parameters of the quantum embedding to increase the kernel-target alignment introduced in Sec. IV.

QEKs have certain appealing properties that make them attractive for use on NISQ devices. As these devices have low coherence times, only quantum circuits of limited depth can be executed. As the computation of the kernel is only a small subroutine in a more general ML technique, it does usually not require long coherence times. Furthermore, the coherence time of the NISQ device can be exploited as much as

* thubregtsen@zedat.fu-berlin.de

possible by tailoring the PQC realizing the QEK to the underlying hardware. Another strong point is that noisy PQCs still lead to well-defined QEKs as will be explained in more detail later. Kernel methods bring with them one main disadvantage: just constructing the kernel matrix already entails quadratic computational complexity in the number of training samples. QEKs are, ultimately, a particular case of kernels, and thus cannot but inherit this scaling, which hinders their application in contexts of large datasets. It is also often hard to choose the *right* kernel for the problem at hand.

This work formalizes and extends our submission to the QHACK 2021 hackathon [24] hosted by Xanadu. We take an educational approach and analyze the whole pipeline necessary to make use of QEKs, including post-processing, error mitigation and optimization of variational parameters. We hope that this work can serve as accessible introduction for interested readers less familiar with QML on near-term devices.

Due to the educational nature of this work, it includes both pre-existing and novel contributions. We propose to use the *kernel-target alignment* as a cost function to train the parameters of the QEK to increase its performance on a particular dataset, as shown in Fig. 1. This technique is closely related to the *metric quantum learning* approach proposed in Ref. [25] for the purpose of optimizing PQCs for the encoding of a specific dataset. Additionally, we discuss the inevitable noise from the underlying device, proposing a mitigation strategy tailored for QEKs that exploits the kernel's definition to infer the underlying noise levels. We also analyze the influence of finite sampling on kernel quality – an issue touched upon in Ref. [21] – showing that the number of samples typically required to achieve an approximation of fixed precision is of third order in the number of datapoints. We review pre-existing strategies that can be used to alleviate the influence of noise on the kernel matrix and propose a different one based on a semi-definite program. We close by providing numerical evidence that kernel training increases classification accuracy and that the proposed noise mitigation methods improve the quality of the obtained kernel matrix.

The rest of this work is organized as follows: In Sec. II and III, we give an intuitive introduction to the theory of kernels in general and quantum embedding kernels in particular. Sec. IV introduces the kernel-target alignment as a measure of fit between dataset and kernel and motivates its use as a cost function for training quantum embedding kernels. In Sec. V, we describe the influence of noise that effects the calculation of kernel matrices in a realistic setting and show how it can be mitigated using knowledge about the ideal kernel matrix. We additionally analyze the influence of finite sampling noise and show how regularization techniques can be used to still ensure that the kernel matrix stays positive semidefinite even. A pipeline for working with QEKs is suggested in Sec. VI. We present results of simulation obtained from both classical and quantum hardware that showcase the proposed approaches in Sec. VII. We conclude by summarizing and discussing both our results and outstanding questions in Sec. VIII.

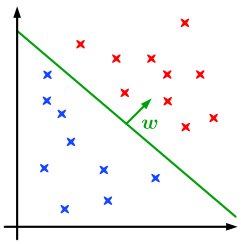

Figure 2. In linear classification, a line (or in higher dimensions a hyperplane) that separates the two classes is sought. We can define the tilt of the line via a vector $\boldsymbol{w}$ orthogonal to it.

II. KERNEL METHODS

Kernel methods are among the cornerstones of machine learning. To understand what a kernel method does, let us first revisit one of the simplest methods to assign binary labels to datapoints: *linear classification*.

Imagine a set of points that lie in different parts of a plane. We want to construct a classifier that successfully predicts the classes of the datapoints. A linear classifier corresponds to drawing a line and assigning different labels $y = \pm 1$ representing the two classes to the opposing sides of the line, as depicted in Fig. 2. Mathematically, this notion can be formalized by introducing a vector $\boldsymbol{w}$ orthogonal to the line, thus determining its direction. We can then assign the class label y to a datapoint $\boldsymbol{x}$ via

$$y(\boldsymbol{x}) = \mathrm{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b), \tag{1}$$

where the intercept term b specifies the position of the line in the plane. The same works for higher dimensional spaces too, where the vector $\boldsymbol{w}$ does not just define a line, but a hyperplane. It is immediately clear that this method is not very powerful, as datasets that are not separable by a hyperplane cannot be classified with high accuracy using this scheme.

There is, however, an ingenious way to enhance the capabilities of a linear classifier: One can specify a *feature map* $\phi(\boldsymbol{x})$ that takes datapoints and embeds them into a larger *feature space* and then perform linear classification in this feature space. In doing so, we can actually realize non-linear classification in the original space of our datapoints. This strategy is visualized in Fig. 3. We can modify the linear classifier of Eq. (1) to include the feature map:

$$y(\boldsymbol{x}) = \mathrm{sgn}(\langle \boldsymbol{w}', \phi(\boldsymbol{x}) \rangle + b), \tag{2}$$

where $\boldsymbol{w}'$ lives in the feature space corresponding to the feature map ϕ.

A major result in kernel theory is the *representer theorem* [26]. It states that one can write the vector $\boldsymbol{w}'$ that defines an optimal decision boundary as a sum of the embedded datapoints with real coefficients: $\boldsymbol{w}' = \sum_i \alpha_i \phi(\boldsymbol{x}_i)$[1]. Inserting

[1] The representer theorem makes mild assumptions about the way we measure "optimal", but for our applications these are always fulfilled.

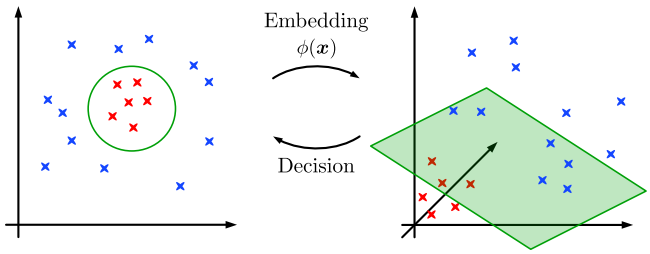

Figure 3. A non-linear embedding can be used to enhance the capabilities of a linear classifier. Linear classification in the embedding space can realize non-linear decision boundaries in the original space.

this into Eq. (2) yields

$$y(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_i \alpha_i \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x})\rangle + b\right). \tag{3}$$

While this might not seem useful at first, notice that the above formula only contains inner products between vectors in the embedding space,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle. \tag{4}$$

We call the function k the *kernel* associated to the feature map ϕ. But why do kernels deserve all the attention they get?

The relevant insight is that we can often find an explicit formula for the kernel k without explicitly performing the feature map ϕ. Consider for example the embedding

$$\phi\colon \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}, \tag{5}$$

whose associated kernel can be explicitly calculated:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{x}') &= \langle \phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \phi \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} \rangle && (6)\\ &= x_1^2 {x'}_1^2 + 2x_1x_2x_1'x_2' + x_2^2 {x'}_2^2 && (7)\\ &= (x_1x_1' + x_2x_2')^2 && (8)\\ &= \langle \boldsymbol{x}, \boldsymbol{x}'\rangle^2. && (9) \end{aligned}$$

We find that it can be obtained by simply computing the inner product of two vectors *in the initial space* and squaring it. Implicitly, we are however computing the inner product relative to the embedding ϕ, i.e., in feature space! This is the central property of kernel-based methods. In many relevant cases the embedding will require a much higher cost to compute than the kernel, while one still gains access to the larger feature space through the kernel. This implicit use of the embedding through its associated kernel is known as the *kernel trick*.

If we do not need the embedding at all, how can we then determine if a given function k is actually a kernel for some feature map? This question is answered by checking the *Mercer condition*, which states that any function fulfilling

$$\sum_{i,j} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0 \tag{10}$$

for *all* possible sets of real coefficients $\{c_i\}$ and sets of datapoints $\{\boldsymbol{x}_i\}$ is a kernel. Alternatively, we can check whether the kernel matrix K with entries

$$K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{11}$$

associated with *any* dataset $\{\boldsymbol{x}_i\}$ is always *positive semi-definite*.

If we now come back to the example of linear classification in a feature space that motivated our introduction to kernel methods, it is natural to ask how we can best choose the separating hyperplane. The most common strategy and application for kernel methods is the *support vector machine* (SVM). The idea behind SVMs is to find the hyperplane with the maximal *margin*. The margin describes the distance of the dataset on either side of the hyperplane. Intuitively, a larger margin is better, since the result would be that outliers of the dataset have a smaller chance of being wrongly classified. This way, SVM is an algorithm that takes as input the kernel matrix from Eq. (11) and delivers the values α_i and b for Eq. (3) that correspond to the decision boundary with the maximal margin.

To predict a class label for a new datapoint, we need to calculate the kernel with respect to the training datapoints and decide on a class label, as shown in Eq. (3). A strong advantage of SVMs is that usually only few weights $\{\alpha_i\}$ contribute significantly to the sum in Eq. (3). We can thus make a prediction by calculating the kernel with respect to these datapoints from the training dataset. The corresponding datapoints are the eponymous *support vectors* – as they support the decision boundary. Intuitively, we can imagine that comparing a new datapoint only to points close to the decision boundary will give important information about the class label.

Kernel methods are not limited to classification. In fact, one can take any ML technique that can be reformulated in terms of inner products and replace the inner products by kernel functions to get a "kernelized" variant. This leads to interesting applications such as kernel principal component analysis [27] or kernel ridge regression [28].

While we have now seen that kernel methods can enhance the power of many ML techniques, the current progress of learning models is driven by deep neural networks, not kernel methods. This is due to their following downsides: For the application of kernel methods, the kernel matrix with respect to the input data needs to be constructed – which has quadratic complexity in the number of datapoints. This can already constitute a substantial impediment in the world of big data where the number of datapoints can be in the millions. Another downside is that the selection of a suitable kernel for a given problem is a non-trivial task. The radial basis function *(RBF) kernel* also known as *Gaussian kernel* given by

$$k_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right) \tag{12}$$

is often a decent starting point, but even there the parameter σ that quantifies how close datapoints need to be to be considered similar needs to be fine-tuned.

Despite the fact that kernel methods do not dominate contemporary ML applications, they are extremely useful to understand learning models in general. This is because many

learning methods can be mapped back to kernel methods for which in turn there exist theoretical guarantees, for example on their capability to generalize on unseen data [26].

III. QUANTUM EMBEDDING KERNELS

On NISQ hardware, we make use of quantum gates, like Pauli rotations, to load data onto the quantum computer. This constitutes a quantum circuit that is represented by a unitary operation dependent on the specific datapoint, $U(\boldsymbol{x})$. As soon as the data is loaded, the quantum system is in a state that depends on the datapoint in question,

$$|\phi(\boldsymbol{x})\rangle = U(\boldsymbol{x})\,|0\rangle\,. \tag{13}$$

This approach is also known as a *quantum feature map* [16] because we are effectively embedding the datapoint in the Hilbert space of quantum states. As we learned in Sec. II, it is no large step from a feature map to a kernel function. We only need to take the inner product between quantum states, which is given by the *overlap*

$$k(\boldsymbol{x}, \boldsymbol{x}') = |\langle\phi(\boldsymbol{x}')|\phi(\boldsymbol{x})\rangle|^2. \tag{14}$$

This is the quantum embedding kernel *(QEK)* associated to the quantum feature map $|\phi(\boldsymbol{x})\rangle$.

In general, we are not able to avoid noise, which means that we cannot assume that the embedded quantum state is pure. In this case, the quantum embedding is realized by a data-dependent density matrix $\rho(\boldsymbol{x})$, which for a pure state reduces to $\rho(\boldsymbol{x}) = |\phi(\boldsymbol{x})\rangle\langle\phi(\boldsymbol{x})|$. The inner product is given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Tr}\{\rho(\boldsymbol{x})\rho(\boldsymbol{x}')\}. \tag{15}$$

This inner product is also known as the Hilbert-Schmidt inner product for matrices. For pure quantum states, this reduces to Eq. (14).

In summary, any quantum feature map induces a QEK. We can use this kernel as a subroutine in a classical kernel method, for example the SVM, which yields a hybrid quantum-classical approach. In this case, the separating hyperplane is constructed in a purely classical manner, only the kernel function between the training datapoints is evaluated on the quantum computer.

To be able to use QEKs in this way, we need to evaluate the overlap of two quantum states on near-term hardware. There are a number of advanced algorithms to estimate the overlap of two quantum states [29–33]. All these algorithms work for arbitrary states, and so they are agnostic to how the states were obtained by necessity. By exploiting the structure and specifics of QEKs, though, we can do better.

For unitary quantum embeddings, i.e. embeddings resulting in a pure quantum state, this is straightforward if we can construct the adjoint of the data-encoding circuit, $U^\dagger(\boldsymbol{x})$. In this case, we can rewrite the overlap as

$$|\langle\phi(\boldsymbol{x}')|\phi(\boldsymbol{x})\rangle|^2 = |\langle 0|U^\dagger(\boldsymbol{x}')U(\boldsymbol{x})|0\rangle|^2. \tag{16}$$

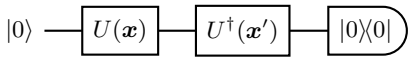

Figure 4. The overlap between the embedded states can be computed by applying the unitary $U(\boldsymbol{x})$ embedding the first datapoint and the adjoint of the unitary embedding the second datapoint $U^\dagger(\boldsymbol{x}')$. This approach results in a doubled circuit depth but does not need auxiliary qubits. It works only for pure states.

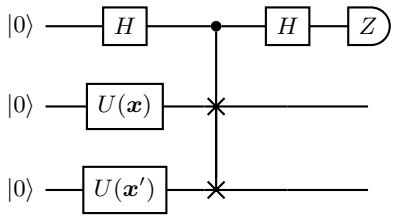

Figure 5. The overlap between the embedded states can be computed by embedding both datapoints in parallel. An auxiliary qubit is then used together with controlled SWAP operations to extract the overlap, which is obtained from the expectation value of the Pauli-Z observable on the auxiliary qubit. This approach results in a doubled circuit width and requires one additional qubit and controlled SWAP operations. It works for pure and mixed states.

This is nothing but the probability of observing the $|0\rangle$ state when measuring the state $U^\dagger(\boldsymbol{x}')U(\boldsymbol{x})|0\rangle$ in the computational basis. In order to obtain an estimate, we can therefore initialize the quantum system in the $|0\rangle$ state, apply the unitary operation $U(\boldsymbol{x})$ followed by $U^\dagger(\boldsymbol{x}')$, and finally measure in the computational basis. From there, we only need to record the frequency with which the prepared state is found in the $|0\rangle$ state to obtain our estimate. The circuit diagram for this *adjoint* approach can be found in Fig. 4. This scheme does not need auxiliary qubits, yet it applies the data-encoding circuit twice (or the adjoint thereof). This way, while the width of the circuit for the adjoint approach does not increase from the data-encoding one, the depth is doubled.

Another alternative approach to estimate the overlap between two quantum states is the *SWAP test*. The SWAP test is based on the *SWAP trick*, a mathematical gimmick that allows us to transform the product of the density matrices in Eq. (15) into a tensor product:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{x}') &= \mathrm{Tr}\{\rho(\boldsymbol{x})\rho(\boldsymbol{x}')\} && (17)\\ &= \mathrm{Tr}\{(\rho(\boldsymbol{x}) \otimes \rho(\boldsymbol{x}'))S\}, && (18)\end{aligned}$$

where S denotes the SWAP operation between the two quantum systems in the states $\rho(\boldsymbol{x})$ and $\rho(\boldsymbol{x}')$. To extract this quantity, the SWAP test makes use of an auxiliary qubit that controls the SWAP operation. The exact circuit is depicted in Fig. 5. While this approach needs auxiliary qubits and a quantum computer roughly twice as wide, its depth increases only ever so slightly, and it also works for mixed quantum states. If the deeper requirements of the adjoint approach were too limiting, the SWAP test would be a natural alternative.

As quantum feature maps occur in many applications of NISQ computing, it is no surprise that QEKs are instrumental beyond their use as subroutines for classical machine learning

methods. In Ref. [22] it was shown that all variational quantum learning methods boil down to kernel methods, providing an opening for kernel theory to explore the properties of these learning models.

IV. TRAINING QUANTUM EMBEDDING KERNELS

Quantum feature maps can have variational parameters. In this section, we discuss how to adjust these parameters to improve the classification capabilities of quantum embedding kernels.

Before we can dive into the details we take a step back to clear up the terminology we use in this section. Adjusting the variational parameters of a quantum feature map – and therefore of its associated QEK – can be seen as a particular instance of *model selection*. For our purposes, we again split this into two steps, first *kernel selection* and second *kernel optimization*.

Kernel selection is choosing a particular quantum circuit *layout*, or *ansatz*, with certain variational parameters. Here, we refer to a *variational family* of kernels, much like the family of RBF kernels defined in Eq. (12) where we vary the standard deviation σ. Kernel optimization corresponds to the process of fixing the variational parameters to ensure a good classification performance on our target data. Both steps are important to assure a good performance of kernel-based methods. In the following, we will shed light on the possible ways to perform kernel optimization.

A straightforward way to perform kernel optimization is *exhaustive parameter search*. For a single parameter $\theta \in \mathbb{R}$ this method consists of fixing a range of parameter values $[\theta_{\min}, \theta_{\max}]$ and an ε-cover of it:

$$\{\theta_{\min}, \theta_{\min} + \varepsilon, \ldots, \theta_{\max} - \varepsilon, \theta_{\max}\}. \tag{19}$$

Then, a SVM is fitted to the training data for every value in the ε-net and we keep the one which attains the highest accuracy score. This way, we need to fit roughly $p := (\theta_{\max} - \theta_{\min})/\varepsilon$ many SVMs in order to pin down the optimal choice of θ. If, on the contrary, our kernel comprised a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^r$, we would still proceed in an analogous way: In a simplified case, we would still fix a range of parameter values $\boldsymbol{\theta} \in [\theta_{\min}, \theta_{\max}]^r$ and an ε-net. Notice how, now, the ε-net would contain approximately p^r sites, since each parameter component θ_i would need to take all possible values between $\theta_{\min}$ and $\theta_{\max}$ for every possible combination of the remaining parameter components! It is thus clear to see how the computational complexity of exhaustive parameter search grows exponentially $\mathcal{O}(\exp(r))$ with respect to the parameter vector dimension r. One particular example of this type is the "leave-one-out" error bound [26], where after fitting the SVM to all points but one, the predicted label for the left out point describes the quality of the kernel. Because of the unfavorable scaling, exhaustive parameter search procedures are thus only suitable for optimizing few parameters.

It is therefore sensible to resort to a proxy quantity that is easier to compute but still acts as a predictor for classification accuracy. Ref. [34] provides an overview of such measures as provided in Refs. [35–40], using the *kernel-target alignment* from Ref. [41] as central building block.

In the following, we assume initially that the training dataset is *balanced*, meaning that it contains equally many datapoints per class. The generalization to unbalanced datasets is straightforward and left to the end of this section.

The central idea behind the kernel-target alignment is that the *labels* for the training set can be seen as an instance of a very particular kernel, which acts as an oracle that always outputs the correct similarity for two datapoints:

$$k^*(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ and } \boldsymbol{x}' \text{ in same class} \\ -1 & \text{if } \boldsymbol{x} \text{ and } \boldsymbol{x}' \text{ in different classes} \end{cases} \tag{20}$$

Of course, in general we do not have access to this *ideal* kernel, but on the training data it is given by the training labels and the kernel matrix predicted by this ideal kernel has entries

$$K^*_{ij} = y_i y_j. \tag{21}$$

This means that, if we place the labels into a vector $\boldsymbol{y}$, we can express the ideal kernel matrix as the outer product of that vector with itself:

$$K^* = \boldsymbol{y}\boldsymbol{y}^T. \tag{22}$$

To get to a measure of how well a kernel captures the nature of the training dataset we need a way to compare the kernel matrix with the ideal one. To obtain the kernel-target alignment we will make use of geometric reasoning. Remember that we can measure the *alignment* of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ by evaluating and normalizing the inner product:

$$\mathrm{A}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\sqrt{\langle \boldsymbol{a}, \boldsymbol{a} \rangle \langle \boldsymbol{b}, \boldsymbol{b} \rangle}}. \tag{23}$$

The alignment is related to the angle $\sphericalangle(\boldsymbol{a}, \boldsymbol{b})$ between the vectors as $\cos \sphericalangle(\boldsymbol{a}, \boldsymbol{b}) = \mathrm{A}(\boldsymbol{a}, \boldsymbol{b})$, which means that the alignment is a quantity that ranges from -1 for vectors pointing in exactly opposite directions to +1 for vectors pointing in exactly the same direction.

We can apply the same reasoning to kernel matrices. To do so, we need to define an inner product between two matrices. For the definition of the kernel-target alignment we will use the *Frobenius inner product*. For that, we simply treat the matrices as if they were column vectors, with every entry of the matrix being a separate entry of the vector. This means that the inner product is just

$$\langle A, B \rangle_F = \sum_{ij} A_{ij} B_{ij} = \mathrm{Tr}\{A^T B\}, \tag{24}$$

from where one defines the alignment of two matrices B and B' as

$$\mathrm{A}(B, B') = \frac{\langle B, B' \rangle_F}{\sqrt{\langle B, B \rangle_F \langle B', B' \rangle_F}}. \tag{25}$$

Now we have all the ingredients to define the kernel-target alignment:

$$\mathrm{TA}(K) = \mathrm{A}(K, K^*) = \frac{\langle K, K^* \rangle_F}{\sqrt{\langle K, K \rangle_F \langle K^*, K^* \rangle_F}}. \tag{26}$$

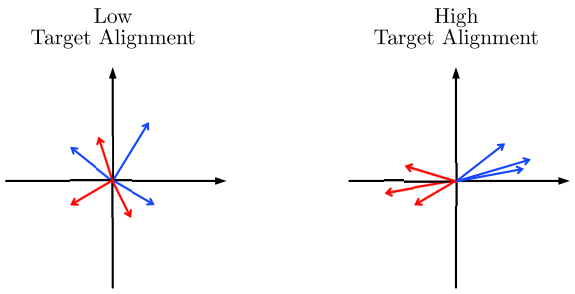

Figure 6. The kernel-target alignment is high if the feature vectors corresponding to datapoints in the same class cluster together and the feature vectors of points in the opposite class lie exactly opposite to them. This illustrates that a high kernel-target alignment allows for easier linear separability.

We can equivalently express this in terms of the kernel function and the training dataset:

$$\mathrm{TA}(k) = \frac{\sum_{ij} y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{\left(\sum_{ij} k(\boldsymbol{x}_i, \boldsymbol{x}_j)^2\right)\left(\sum_{ij} y_i^2 y_j^2\right)}} \tag{27}$$

$$= \frac{\sum_{ij} y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)}{n\sqrt{\sum_{ij} k(\boldsymbol{x}_i, \boldsymbol{x}_j)^2}}, \tag{28}$$

where we used the fact that for all labels $y_i^2 = 1$ and n denotes the number of points in the training set.

At the beginning of this section we assumed that the training set was balanced, i.e., that it contains the same number of datapoints for each class. If this were not the case, the approach we just outlined would run into problems, because the contributions from one class would dominate the kernel-target alignment. We could however mitigate this by simply rescaling the labels, dividing them by the number of samples available in their class. In this case, we cannot use Eq. (28) but have to stay with Eq. (27).

We can gather further intuition why the kernel-target alignment is a meaningful measure by looking at the numerator of Eq. (27), $\sum_{ij} y_i y_j\, k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This quantity is also known as the *kernel polarity*. Each term $y_i y_j\, k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the sum is a product of the kernel function of two points and their labels. If both points belong to the same class, $y_i y_j = +1$, the kernel value increases the kernel-target alignment, whereas if the labels are different $y_i y_j = -1$, the term decreases it. Increasing the kernel-target alignment therefore means both increasing the kernel values for datapoints from the same class and decreasing them for datapoints in different classes. Fig. 6 visually illustrates how a large kernel-target alignment allows for easier linear classification of the training data. Beyond this intuition, the kernel-target alignment profits from theoretical guarantees regarding both its high concentration about the expected value and its good generalization behavior in labeling previously unseen data [40–42].

With the kernel-target alignment we now have a measure that we can use as a cost function to maximize through a hybrid quantum-classical optimization loop [43]. At every iteration, the *quantum processing unit* (QPU) is used to evaluate the kernel matrix K, recall constructing K has quadratic complexity in the number of datapoints.

It turns out that optimizing the kernel-target alignment is closely related to the "quantum metric learning" approach put forward in Ref. [25] that analyzed different strategies to optimize quantum feature maps. Indeed, the Hilbert-Schmidt distance-based method is the same as optimizing the unnormalized kernel-target alignment, the polarity. We detail the connection in App. A.

V. THE EFFECTS OF NOISE

Noise is one of the namesakes of NISQ devices and considering the effects of noise is therefore of utmost importance. In this section, we discuss how both noise arising from imperfect quantum operations – device noise – and noise arising from finite sampling of expectation values affects QEKs and how it can be mitigated.

A. Device Noise

NISQ devices suffer from unavoidable noise caused by unintentional interactions with the environment or imperfect control. It is thus not possible to prepare pure quantum states with an embedding circuit. This fact has multiple implications for QEKs and their implementation.

Noise can be modeled by *quantum channels*. Formally, any map that takes valid quantum states to valid quantum states can be seen as a quantum channel. An example is *depolarizing noise*, which corresponds to a complete loss of information about the underlying quantum state with a certain probability $1 - \lambda$. We formally realize it by replacing the system's quantum state with the maximally mixed state with probability $1 - \lambda$:

$$\mathcal{D}_\lambda[\rho] = \lambda\rho + (1-\lambda)\frac{1}{2^N}. \tag{29}$$

Depolarizing noise is a popular model for noise in quantum systems, as it is simple and subsumes other, more nuanced noise models.

As we have seen in Sec. III, the QEKs can also be defined for mixed states for which it corresponds to the state overlap. The SWAP test can be directly employed to compute the overlap of mixed states (see Fig. 5), but often we would like to use the adjoint method due to its lower qubit requirements (see Fig. 4). The adjoint method, however, needs more consideration, because the implementation of the adjoint noisy embedding circuit itself is not straightforward. But if we fail to implement the correct adjoint operation, we are no longer computing the overlap of the quantum embedding states and therefore also do not compute a valid kernel!

In the noiseless embedding circuit, all operations are unitary and typically the adjoint of every elementary operation is available. This becomes apparent when we consider that

any quantum circuit can be constructed from controlled NOT gates, which are self-adjoint, and single-qubit Pauli rotations, whose adjoint is obtained by performing the same rotation with negated angle.

The device noise, however, can in principle prevent us from implementing the adjoint embedding. As an example, consider a quantum channel that represents a noisy Pauli rotation gate, $\mathcal{V}(\theta)$. We model it by the original rotation gate $R(\theta)$ that is followed by a noise channel $\mathcal{N}$. The channel $\mathcal{N}$ could model imprecision in the control of the rotation angle, unwanted interactions with the environment or other noise processes, but we will leave it arbitrary for the sake of the example. The noisy gate is then given by

$$\mathcal{V}(\theta)[\rho] = \mathcal{N}[R(\theta)\rho R^\dagger(\theta)]. \tag{30}$$

The adjoint of the noisy Pauli rotation gate is then given by

$$\mathcal{V}(\theta)^\dagger[\rho] = R(-\theta)\mathcal{N}^\dagger[\rho]R^\dagger(-\theta). \tag{31}$$

But how would we implement $\mathcal{V}(\theta)^\dagger$? The very nature of a noise channel implies that we cannot control it or choose at which time the noise occurs. Instead, we have to work with the noisy quantum gates at our disposal, which means that we can only approximate $\mathcal{V}(\theta)^\dagger$ by $\mathcal{V}(-\theta)$:

$$\mathcal{V}(-\theta)[\rho] = \mathcal{N}[R(-\theta)\rho R^\dagger(-\theta)] \neq \mathcal{V}(\theta)^\dagger. \tag{32}$$

In general, this approximation is not equal to the adjoint of the noisy unitary. This only happens if the noise channel $\mathcal{N}$ is self-adjoint and commutes with the unitary operation $R(\theta)$, as is true for the depolarizing noise introduced in Eq. (29).

Let us now take a step back and look at actual NISQ devices. They are usually programmed at the gate-level, assuming perfect unitaries. The adjoint of a perfect unitary circuit is readily available, but only if the behavior of the available NISQ device is well-modeled by depolarizing noise can we expect this "naive" adjoining of the unitary gates to still compute the overlap of embedded states for the QEK.

B. Mitigating Depolarizing Noise

Mitigating the effects of device noise is very important to make NISQ practice. It is therefore no surprise that the topic has gained a lot of attention and that many techniques have been developed to mitigate device noise [44–50]. In the following, we will complement these with an approach that exploits the very definition of the quantum embedding kernel and that can be freely combined with other mitigation approaches.

We have introduced depolarizing noise as a rather general approach to model the noise in quantum devices. We will model the noise with $\mathcal{D}_\lambda$ as in Eq. (29), where the depolarizing channel is assumed to act homogeneously on the whole system. We will refer to λ – the probability that the depolarizing channel does not cause a loss of information about the underlying state – as *survival probability*. Note that it may well be possible that the probabilities λ_i differ for distinct embedded datapoints $\boldsymbol{x}_i$, as one might need longer pulse sequences to be embedded than the other, causing more noise.

We now assume that the embedding is composed of this noise channel and the noiseless unitary embedding:

$$\rho(\boldsymbol{x}) = \mathcal{D}_\lambda[|\phi(\boldsymbol{x})\rangle\langle\phi(\boldsymbol{x})|]. \tag{33}$$

We can exploit the composition rule $\mathcal{D}_{\lambda_1}\mathcal{D}_{\lambda_2} = \mathcal{D}_{\lambda_1\lambda_2}$ to compute the noisy kernel matrix entries

$$\begin{aligned} K_{ij}^{(\mathrm{dev})} &= \mathrm{Tr}\{\rho(\boldsymbol{x}_i)\rho(\boldsymbol{x}_j)\} && (34)\\ &= \mathrm{Tr}\{\mathcal{D}_{\lambda_i}[|\phi(\boldsymbol{x}_i)\rangle\langle\phi(\boldsymbol{x}_i)|]\mathcal{D}_{\lambda_j}[|\phi(\boldsymbol{x}_j)\rangle\langle\phi(\boldsymbol{x}_j)|]\} && (35)\\ &= \mathrm{Tr}\{\mathcal{D}_{\lambda_i\lambda_j}[|\phi(\boldsymbol{x}_i)\rangle\langle\phi(\boldsymbol{x}_i)|\phi(\boldsymbol{x}_j)\rangle\langle\phi(\boldsymbol{x}_j)|]\} && (36)\\ &= \lambda_i\lambda_j K_{ij} + (1-\lambda_i\lambda_j)\frac{1}{2^N}. && (37) \end{aligned}$$

We can exploit the fact that all diagonal entries of the noiseless kernel matrix K are known to be 1. While we could use this knowledge to save quantum computational cost, we propose to use it to gather information about the device noise. We can use Eq. (37) to infer the survival probability λ_i from the diagonal element of the noisy kernel matrix $K_{ii}^{(\mathrm{dev})}$:

$$\lambda_i = \sqrt{\frac{K_{ii}^{(\mathrm{dev})} - 2^{-N}}{1-2^{-N}}}. \tag{38}$$

With those values at hand we can recover the noiseless kernel matrix entries

$$K_{ij} = \frac{K_{ij}^{(\mathrm{dev})} - 2^{-N}(1-\lambda_i\lambda_j)}{\lambda_i\lambda_j}. \tag{39}$$

We denote this mitigation strategy as M-SPLIT. We can deduce two even simpler mitigation strategies from this approach by assuming that all λ_i have the same value. This value can be estimated by averaging multiple of the λ_i obtained from Eq. (38), a strategy which we denote as M-MEAN and which requires less diagonal elements to be measured. Alternatively, we can choose to further save resources and only measure one diagonal entry to estimate the survival probability, which we denote as M-SINGLE. There are a number of options for which entry to use, for the sake of simplicity and reproducibility we always use the first entry.

C. Finite Sampling Noise

Recall Eq. (14), where we first introduced the definition of QEKs. And, critically, notice from Eq. (16) that we proposed the so-called adjoint method for estimating the overlap: a frequentist way of approximating the kernel function from quantum circuit evaluations. Measuring such kernel functions results in *independent and identically distributed* (i.i.d) Bernoulli random variables $\hat{k}_{ij}$, since each circuit evaluation outputs either a 1, in case the observed state is $|0\rangle$, or a 0 otherwise. By construction, the theoretical kernel value K_{ij} is the

true mean of this random variable, i.e., $\mathbb{E}(\hat{k}_{ij}) = K_{ij}$. Since it follows from Born's rule that an infinite number of circuit evaluations would be required to pin down the exact kernel value, there exists a second source of noise originating from using a finite number of samples.

In reality, we can only estimate the kernel function from a finite number of experimental runs. How many runs we can afford is limited by our experimental resources. This incurs uncertainty beyond the device noise, especially if the number of runs is small.

Going one step further, notice that the pipeline involves estimating the entire kernel matrix, comprising $n(n-1)/2 = \mathcal{O}(n^2)$ independent entries. To gauge the number of required circuit evaluations M_{tot} to reach a desired error ϵ in operator distance of an estimator to the target kernel matrix, we can use results from random matrix theory.

The difference between an estimator constructed using M circuit evaluations *per entry*, $(\bar{K}_M)_{ij} = \sum_{s=1}^{M} \hat{k}_{ij}^{(s)}/M$, and the target kernel matrix K is given by

$$(\bar{E}_M)_{ij} = (\bar{K}_M)_{ij} - (K)_{ij} = \frac{1}{M}\sum_{s=1}^{M} \hat{k}_{ij}^{(s)} - K_{ij}. \quad (40)$$

Remembering that

$$\mathbb{E}\left\{(\bar{E}_M)_{ij}\right\} = 0, \quad (41)$$

$$\mathbb{E}\left\{[(\bar{E}_M)_{ij}]^2\right\} = \mathcal{O}\left(\frac{1}{M}\right), \quad (42)$$

$$\mathbb{E}\left\{[(\bar{E}_M)_{ij}]^4\right\} = \mathcal{O}\left(\frac{1}{M^2}\right), \quad (43)$$

allows us to make use of the following result:

Theorem 1 (Latala's theorem [51, 52])**.** *Let A be a random matrix whose entries a_{ij} are independent centered random variables with finite fourth moment. Then, for $C > 0$,*[2]

$$\begin{aligned}\mathbb{E}\{\|A\|\} \leq C\Big[&\max_i \big(\sum_j \mathbb{E}(a_{ij}^2)\big)^{1/2} \\ &+ \max_j \big(\sum_i \mathbb{E}(a_{ij}^2)\big)^{1/2} \\ &+ \big(\sum_{ij} \mathbb{E}(a_{ij}^4)\big)^{1/4}\Big].\end{aligned} \quad (44)$$

For our $n \times n$-dimensional error matrix $\bar{E}_M$ with moments as in Eqs. (41-43), this leads to

$$\mathbb{E}\{\|\bar{E}_M\|\} = \mathcal{O}\left(\frac{\sqrt{n}}{\sqrt{M}}\right). \quad (45)$$

Consequently, $M = \mathcal{O}(n/\epsilon^2)$ measurements per kernel matrix entry are required to ensure an error of ε in operator distance. As a result, since we need to estimate $\mathcal{O}(n^2)$ entries of the kernel matrix, we require a total of

$$M_{\text{tot}} = \mathcal{O}\left(\frac{n^3}{\epsilon^2}\right) \quad (46)$$

circuit evaluations to reach the desired accuracy. A short calculation for Gaussian variables using Bai-Yin's law [52, 53] verifies that this scaling is indeed asymptotically optimal, since the error of kernel matrix entries converges to the Gaussian distribution according to the central limit theorem.

The corresponding constant prefactors can be obtained via involved methods using results from random matrix theory for matrices with subgaussian rows [52]

Another important quantity we need to estimate for quantum embedding kernels is the kernel target alignment introduced in Eq. (27), which we use during the kernel training. Allowing for an error of ε in the kernel target alignment, error propagation suggests that $\mathcal{O}(1/\varepsilon^2)$ measurements per kernel entry are required, leading to $\mathcal{O}(n^2/\epsilon^2)$ circuit evaluations in total.

D. Regularizing the Kernel Matrix

Due to the imperfect sampling outcome for the kernel matrix and the device noise mitigation techniques introduced above, the obtained matrix might not be positive semi-definite. However, we know the exact kernel matrix to be positive semi-definite and this property is a requirement for the matrix to be used in a classification task. We may therefore regularize the obtained matrix, validating it as kernel matrix and bringing it closer to the perfect outcome.

We discuss three methods to find a positive semi-definite matrix close to a symmetric matrix A: In the first method called *Tikhonov regularization*, we displace the spectrum of A by its smallest eigenvalue $\sigma_{\min}$ if it is negative, by subtracting it from all eigenvalues or equivalently from the diagonal [54]:

$$\mathsf{R\text{-}TIK}(A) = \begin{cases} A - \sigma_{\min}\mathbb{1} & \text{if } \sigma_{\min} < 0 \\ A & \text{else} \end{cases}, \quad (47)$$

which yields a positive semi-definite matrix. While being formally the same as the original method by Tikhonov [55], we use it here to assure positive semi-definiteness instead of non-singularity of the matrix.

The second method called *thresholding* only changes the negative eigenvalues of A by setting them to zero [56]. This is done via a full eigenvalue decomposition, adjustment of the negative eigenvalues and composition of the adjusted spectrum and the original eigenvectors:

$$D = V^T A V \quad (48)$$

$$D'_{ij} = \max\{D_{ij}, 0\} \quad (49)$$

$$\mathsf{R\text{-}THR}(A) = V D' V^T. \quad (50)$$

This approach is equivalent to finding the positive semi-definite matrix closest to A in any unitarily invariant norm. It is also equivalent to finding the positive semi-definite matrix which has the largest alignment (see Eq. (26)) with A.

[2] C is a constant depending only on the subgaussian norm of the entries.

The third method extends this reasoning by searching the closest matrix in Frobenius norm, but with the additional requirement that the diagonal elements of the regularized matrix need be one, incorporating our knowledge about the exact kernel as a constraint. This approach constitutes an *semi-definite program* (SDP) and is therefore efficiently computable:

$$\mathsf{R\text{-}SDP}(A) = \operatorname{argmin}\left\{\|A' - A\|_F : A' \succcurlyeq 0,\ A'_{ii} = 1\right\}. \tag{51}$$

For further details on the computational cost and properties of the regularized matrices please refer to App. B 1.

We note that other mitigation techniques proposed in the literature may be combined with the ones discussed here, as they function on different levels of abstraction. An established method to reduce the impact of noise is zero noise interpolation to first order [15, 44, 57], which makes use of additional circuit evaluations at increased noise rates. Furthermore, techniques to suppress errors by duplicating the circuit have been proposed recently [58, 59]. Both of these methods may be used to greatly reduce the noise on the kernel matrix before treating it with regularization and mitigation techniques.

During the preparation of this work, Wang *et al.* [60] demonstrated that regularization methods can improve the classification accuracy of noisy circuits significantly, more concretely R-TIK, R-THR and flipping the negative eigenvalues of the kernel matrix were covered.

VI. QEK PIPELINE

The approaches presented in the previous sections can be combined into a holistic pipeline for working with quantum embedding kernels as depicted in Fig. 7. We start from a parameterized quantum circuit that represents a quantum feature map with variational parameters set to some initial values. To adjust the parameters for a specific dataset, a training loop is run: First, a kernel matrix is obtained from the underlying NISQ device. As an alternative next step, a mitigation and regularization strategy can be applied to improve the quality of the kernel matrix. The kernel matrix is then used to calculate the kernel-target alignment and its gradient with respect to the variational parameters of the parameterized quantum circuit. Gradient descent is then used to update the variational parameters and hence the quantum embedding kernel. This process is repeated until the desired kernel-target alignment is reached.

To perform a classification of new data, a support vector machine is trained using the post-processed kernel matrix of the optimized quantum embedding kernel. The support vectors of the SVM are then extracted and can be used in a support vector classifier to predict labels for new datapoints. To this end, the quantum embedding kernel between the new datapoints and the support vectors have to be computed, but the training of the SVM itself is purely classical.

VII. NUMERICAL EXPERIMENTS

Let us make our discussion concrete by designing and performing some proof-of-principle experiments. We consider both noiseless and noisy simulations of quantum circuits as well as experiments on actual NISQ devices. Each of our experiments is designed to illustrate a specific step of the main pipeline depicted in Fig. 7. Accordingly, we use different datasets and QEKs curated to each experiment and repetition.

For the sake of illustration we choose to work on datasets of 2 dimensions. We use two artificial and one semi-artificial dataset. The exact construction of each dataset is detailed in App. C.

The *checkerboard* dataset represents a 4×4 grid of alternating classes, where the elements of the checkerboard are drawn from a continuous uniform distribution centered in the tiles of the checkerboard. Due to its many different connected components, the checkerboard dataset requires a medium sized QEK, which we use for benchmarking our mitigation and regularization techniques. Next, we sample pixels from MNIST handwritten digit images of classes 0 and 1 and generate datasets *zero vs. not-zero*, *one vs. not-one*, *base-zero* and *base-one* to show a more realistic use of QEKs, for which we use much larger embedding circuits. Finally, we have the *symmetric donuts* dataset, consisting of two pairs of circumscribed circles with alternating classes. We use real hardware for this dataset and thus use a fairly shallow and narrow QEK.

For our experiments, we always used the adjoint approach because of its lower qubit requirements. For the quantum feature map, we follow Refs. [14, 61, 62] and use a *data reuploading* quantum embedding circuit.

The circuit used in this work is a repetition of the following layer architecture that is responsible for both embedding data and applying trainable gates. One layer comprises: one layer of Hadamard gates; one layer of single-qubit Pauli-Z rotations, embedding one data feature each; one layer of trainable single-qubit Pauli-Y rotations; and one *ring* of controlled Pauli-Z rotations, in which each qubit acts as control for the rotation on the next adjacent qubit, considering first and last as adjacent. Note that all controlled Pauli-Z rotations mutually commute, therefore allow to execute the ring in constant circuit depth. Consequently, for a circuit constructed using L blocks and N qubits, there are $2NL$ trainable parameters.

Furthermore, we do not tie the i^{th} feature to one particular qubit, but rather iterate over the features cyclically in each embedding layer. That means, if there are more data features n than qubits in the circuit N, one block is not enough to encode all features. In this case, we encode the first $x_1, \ldots, x_N$ features in the first block, and for the second block we start encoding features x_{N+1}, x_{N+2}, and so on until we reach x_n. From there on, provided there are still more encoding gates available in the circuit, we start over from x_1. The circuit diagram for a single block encoding 2-dimensional data on 5 qubits is shown in Fig. 8.

When training our models, we treat the number of qubits N and the number of blocks L as hyperparameters of the problem, much like the number of neurons per layer and the number of layers in artificial neural networks. In QML, the num-

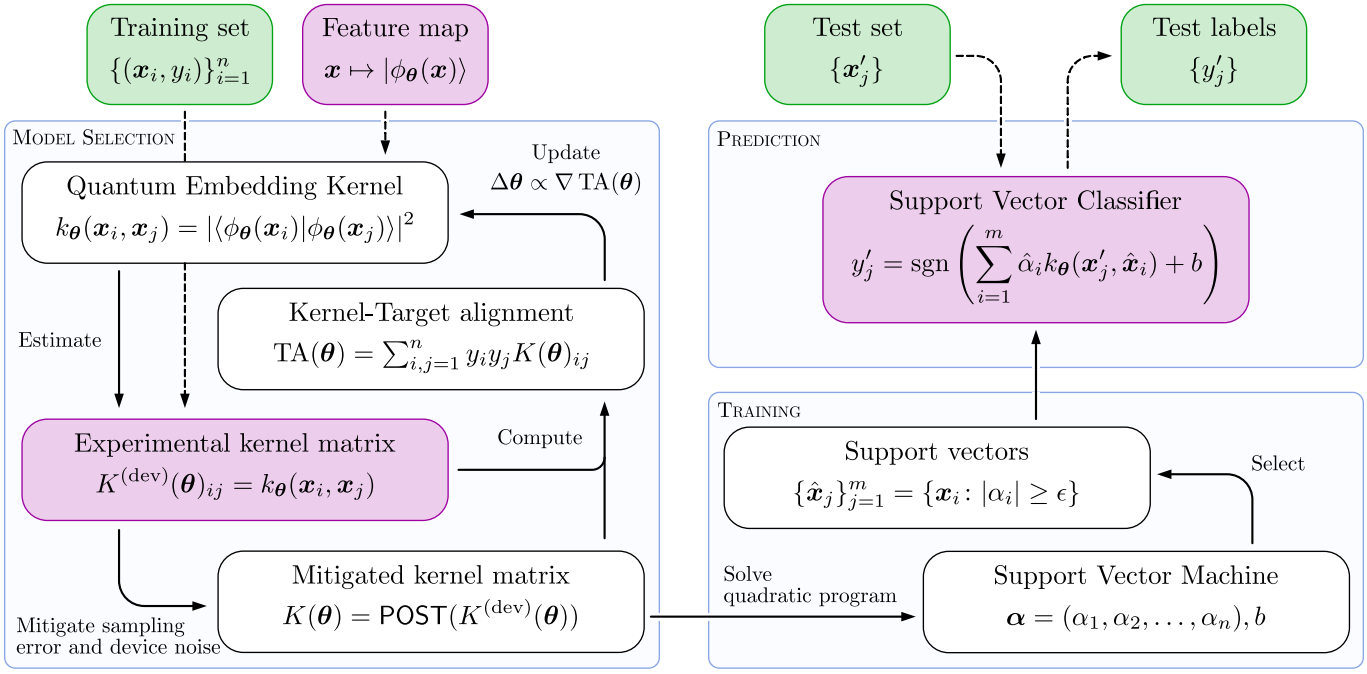

Figure 7. Schematic of the pipeline used in this work. Green boxes indicate data, purple boxes indicate process steps that are executed on quantum hardware. The pipeline used in this work can be split into three separate parts. In the model selection part, depicted on the left, the parameters of the feature map are adjusted to increase the kernel-target alignment. To calculate the alignment, the kernel matrix is computed and may afterwards be post-processed to mitigate sampling and device noise. After a sufficient target-alignment is reached, the kernel is used to train a support vector machine. The resulting support vector classifier is used in the prediction step to predict labels of new datapoints.

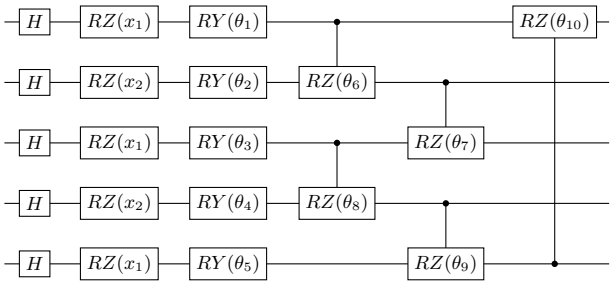

Figure 8. Circuit diagram of the elementary building block used in the QEK ansatz for $N = 5$ qubits and $m = 2$ features. The trainable parameters of the ansatz are denotes as $\{\theta_j\}_{j=1}^{s=10}$ and $x_{1,2}$ are the data features.

ber of qubits is usually referred to as *width*, and the number of blocks as *depth*.

With these kernels, we perform three experiments, corresponding to the three sections that follow. In Sec. VII A we check the validity of training QEK via noiseless simulations. Next, we allow for noise sources (as considered in Sec. V) to test the aforementioned mitigation and regularization techniques in Sec. VII B, still on a classical simulator. Finally, the culminating experiment studies mitigation techniques again, but this time using IonQ's QPU [63], to demonstrate functionality on an actual quantum device.

Our experiments were implemented using the PennyLane library for QML [64] and code is available at [24]. Hardware experiments were executed on an IonQ device [63] via Amazon Braket [65].

A. Noiseless simulations

We kick off by testing our approach on ideal lab conditions. As a first step, we assume both completely negligible noise and access to infinite samples. This can be achieved using a classical simulator that stores the entire wave vector at every step. We perform several experiment repetitions using all introduced datasets.

The experiment takes three inputs: the hyperparameter values for the QEK, the training set, and the test set. Once these are provided, the steps are always: (1) sample a few sets of kernel parameters at random (we do 3 or 5). We refer to the kernels using these sets of parameters as *untrained* kernels. (2) to each untrained kernel, fit a SVM using the training set. Denote the resulting classification accuracy on test as *untrained* accuracy. (3) select one of the untrained kernels based on some quality of its respective accuracy (we select the lowest one). (4) tune the parameters of the selected untrained kernel by maximizing the kernel target alignment (we use for instance Stochastic Gradient Ascent, with a batch size as small as 4). Once optimization is finished, we call the resulting object the *trained* kernel. (5) fit one last SVM using the trained kernel and the training set and report the achieved accuracy on the test data. This is the *trained* accuracy. Of

our interest in this experiment are, as said, the untrained and trained accuracy.

There is one additional experiment designed specifically for the semi-artificial MNIST dataset. For semi-artificial MNIST, we train the kernels only on the zero vs. non-zero and one vs. non-one datasets. We tackle a "Zero vs. one" membership problem with the base-zero and base-one sets via a majority vote ensemble of the trained kernels from last step. More extensive explanation on the ensemble experiment to be found in App. C.

According to the theory presented in Sec. IV and in [34], one should expect that a trained QEK should outperform the randomly initialized kernel pre-training[3]. We expect that larger QEK profit more from training, because deeper and wider systems have a larger parameter space, and random initialization therefore has a lower chance of falling close to a local maximum. More trainable parameters also convey more expressivity.

Fig. 9 shows how the quality of the decision boundaries improves from the untrained kernels (left column) to the trained ones (right column). In all experimental repetitions, though, there are still some misclassified points, which is not unexpected as the underlying kernel functions likely have limited expressivity. The precise numerical results of this experiment are found in Tab. I. The table lists the minimum and maximum untrained accuracy with random parameters for the kernel, the trained accuracy and the choice of hyperparameters for each experiment repetition. Notice that the "zero vs. one" ensemble experiment does not have untrained kernels by construction.

Upon inspecting the results, we notice how trained accuracy values range from 0.75 for the smallest QEK (width and depth equal to 3) to 0.97 for repetitions with larger circuits and we even obtain a perfect score for the ensemble experiment. We identify the anticipated general trends: training the kernel always improved the accuracy compared to the untrained kernel with minimum accuracy in our experiment. This is empirical evidence that maximizing kernel target alignment comes with improved accuracy. Despite the consistent improvement over the untrained kernel with minimum accuracy, the untrained kernel with maximum accuracy matches the trained kernel in the "zero vs. not-zero" repetition, and even comes out on top for the smaller instance of "symmetric donuts". Note that increasing the size (and implicitly the expressivity) of the embedding circuit in the "symmetric donuts" repetition leads to the *trained* QEK accuracy being higher than the maximum *untrained* one.

[3] Important noting is that target alignment optimization is a heuristic that aims at guaranteeing *good* classification and generalization, not *better*. There exist adversarial datasets where perfect accuracy can be achieved with very low alignment. For instance, consider a dataset where points of each class are organized in two long parallel lines, one label per line.

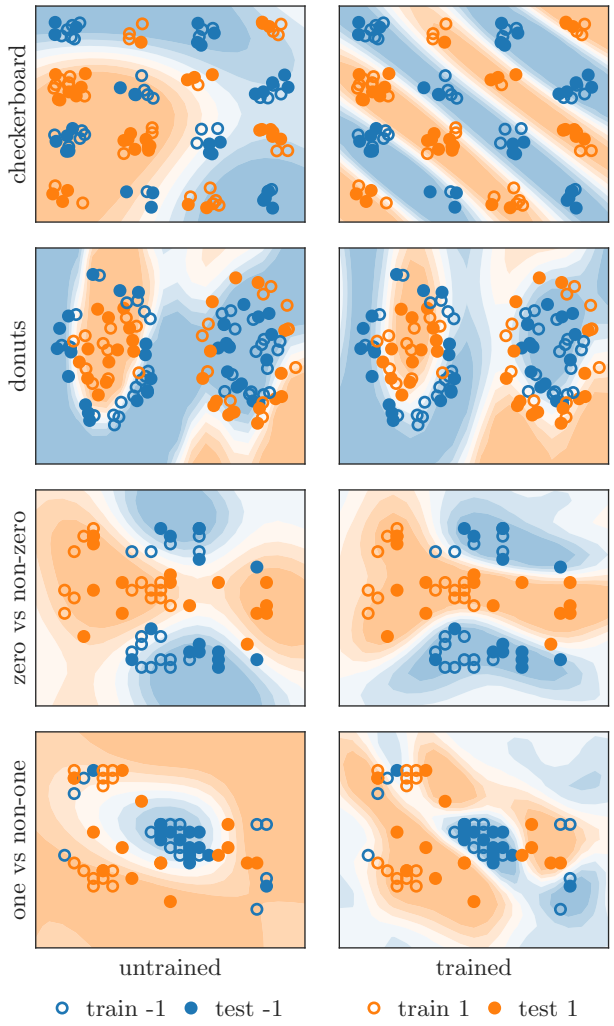

Figure 9. Datasets and decision boundaries of SVMs constructed using different QEKs in noiseless simulations. Each plot shows the real plane and the axes are cartesian coordinates in arbitrary units. The rows correspond to the different datasets, of which a detailed account can be found in App. C. The columns correspond to the kernel before (left) and after (right) target alignment training. The classification accuracy achieved in every repetition is presented in Tab. I, next to the respective hyperparameters.

B. Mitigation and regularization experiments

We now investigate the effect of the regularization and device noise mitigation techniques introduced in Sec. V B and V D. To this end, we simulate device noise with a noise model based on local depolarizing noise (see App. C 5 for details) and test the post-processing performance on the checkerboard dataset.

For a range of base survival probabilities λ_0 and measurements M per kernel matrix entry, we first compute the noisy, sampled kernel matrix $\bar{K}_M$. We then consider any combination of up to three methods with the order *regularize - mitigate - regularize*, including combinations that skip one or two

Table I. Kernel experiments and ensemble experiments - datasets, circuit hyperparameters and achieved accuracies

Dataset	total samples (n)	Width (N)	Depth (L)	Untrained (minimum)	Untrained (maximum)	Trained (from minimum)
Checkerboard	60 datapoints	5	8	0.52	0.52	0.97
Zero vs not-zero	30 datapoints	4	32	0.9	0.97	0.97
One vs not-one	30 datapoints	4	32	0.77	0.77	0.97
Zero vs one	10 images (15 datapoints each)	4	32	NA	NA	1
Symmetric Donuts	120 datapoints	3	3	0.58	0.82	0.75
Symmetric Donuts	120 datapoints	4	3	0.65	0.70	0.85

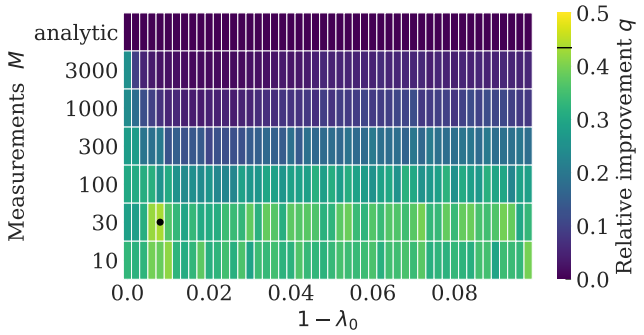

Figure 10. Relative improvement q in alignment (see Eq. (52)) for several base survival probability λ_0 and numbers of circuit evaluations per matrix entry M under a noise model based on local depolarizing noise (see App. C 5). We here use the best of our post-processing combinations, which consists only of R-SDP, rated by how often they performed best across all noise regimes (for the detailed ranking see App. B 2). The additional line in the color bar and the marker in the heatmap indicate the best relative improvement and the corresponding noise parameters.

of these steps, to post-process $\bar{K}_M$ into $K^{(\mathrm{post})}$. The quality of each mitigation strategy is finally determined as the change in alignment with K (introduced in Eq. (25)), the noiseless matrix, compared to $\bar{K}_M$, relative to the deviation of the raw matrix from perfect alignment:

$$q = \frac{\mathrm{A}\big(K^{(\mathrm{post})}\big) - \mathrm{A}\big(\bar{K}_M\big)}{1 - \mathrm{A}\big(\bar{K}_M\big)}, \tag{52}$$

where in a slight abuse of notation we abbreviated $\mathrm{A}\big(K^{(\mathrm{post})}\big) := \mathrm{A}\big(K^{(\mathrm{post})}, K\big)$ and skipped the dependence on the exact kernel matrix K.

Fig. 10 shows the improvement q across the base survival probabilities and numbers of circuit evaluations for the best of the 42 combinations of mitigation and regularization strategies, achieving values between 0% and 43.5%. We especially see larger improvements for lower numbers of measurements. For more details on the considered combinations of regularization and mitigation and for the best choice for different noise regimes, please refer to App. B 2.

Overall, we see that post-processing can in general enhance the quality of the obtained kernel matrix significantly, in addition to other possible mitigation techniques that may reduce the sampling and device noise strengths effectively [15, 44, 57] or even at the hardware level [58, 59]. At the same time, one has to choose the mitigation and regularization techniques carefully as their quality depends on the number of circuit evaluations and device noise level. In particular, the post-processing methods are not attributed to one source of noise each, but e.g. regularization might be the better strategy for tackling device noise, which it was not designed to counter, as we saw in the regime of strong device noise. If no estimate for the noise levels is available, applying a simple regularization routine such as thresholding offers systematic and consistent improvement, but for realistic noise levels, device noise mitigation typically provides even better results (also see Sec. VII C and App. B).

When using one of the mitigation techniques, one should consider that for M-SINGLE, only one of the diagonal entries of the kernel matrix needs to be calculated, while multiple (all) diagonal entries are required for M-MEAN (M-SPLIT).

C. Hardware experiments

So far, all experiments in this section were run on classical simulators. While we tried to use fair noise models and reasonable circuits, it is worthwhile to investigate the behavior in real-world conditions. To test the performance of the introduced techniques on actual quantum hardware, we have computed the kernel matrix for the symmetric donuts dataset using three qubits on an ion trap QPU by IonQ.

For the computation we have used $M = 175$ circuit evaluations per kernel matrix entry and because we measured the diagonal entries for mitigation purposes, the total number of circuit evaluations sums up to about $3.2 \cdot 10^5$ in total. In addition, we have sampled kernel matrices for several smaller M values from the measured distribution[4].

Fig. 11 shows the alignment $\mathrm{A}\big(\bar{K}_M\big)$ between the obtained kernel matrix $\bar{K}_M$ and the noiseless matrix K, as well as the alignment $\mathrm{A}\big(K^{(\mathrm{post})}\big)$ between the post-processed matrix $K^{(\mathrm{post})}$ and K. For each number of circuit evaluations M, we plot the two best of the 42 post-processing combinations. Note that many of these combinations yield a quality similar

[4] Note that this is not the same as a proper computation on the quantum device with decreased M because we sample from a sample and not from the true distribution directly.

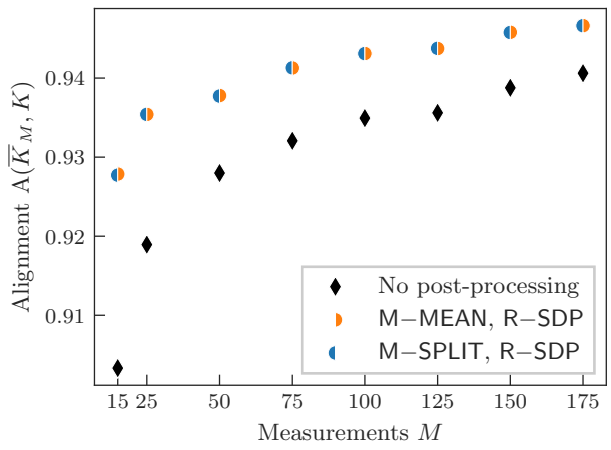

Figure 11. Alignment A of the kernel matrix measured on the ion trap QPU with the simulated, noiseless kernel matrix K for various numbers of circuit evaluations per matrix entry M, with and without the respective two best post-processing strategies. Applying our device noise mitigation techniques M-MEAN/M-SPLIT (see Sec. V B), which assume a simple, global depolarizing noise model, and matrix regularization R-SDP results in an improved alignment.

to the best choice. As expected, the quality of the kernel matrix improves with the number of circuit evaluations and as predicted by our simulation results (see App. VII B), the post-processing methods increase the alignment significantly. The achieved values for the relative improvement q range between 10.1% and 25.4% with a mean of 14.9%.

We note that the combination M-MEAN, R-SDP, which is either best or second best in the hardware results, was correctly predicted for small device noise levels by our simulations of depolarizing noise, on a different dataset, with different circuit depth and width (see App. B) and compiled to a different elementary gate set. This indicates that the depolarizing noise model captures properties of the noise in the QPU that are significant for the kernel matrix computation, and suggests that these post-processing methods show robust performance across different circuit depths, qubit numbers and datasets.

In conclusion, our results on the actual quantum device demonstrate an increased kernel matrix quality when using post-processing, which may allow for improved classification accuracy (also see [60]) or alternatively for a reduced number of circuit evaluations while maintaining a fixed classification performance.

VIII. SUMMARY AND OUTLOOK

In this work, we have studied the concept of *quantum embedding kernel* (QEK). Here, quantum embedding circuits serve as feature maps for kernel-based machine learning (ML). To optimize variational parameters of the QEKs, we transferred the concept of kernel target alignment to the quantum setting. We have performed various numerical experiments that showed improvement in classification accuracy after training.

As the kernel matrices must be positive semi-definite, and the QEKs run on near-term quantum device, we have also investigated noise mitigation techniques. Concretely, we proposed device noise mitigation techniques specific for kernel matrices and combined them with regularization methods. We tested a large set of combinations, both on simulated depolarizing noise as well as on data from a real quantum processing unit. In both scenarios we found that post-processing methods can partly recover the noiseless kernel matrix. Based on these results we recommend best-practice post-processing strategies for different noise regimes.

There are two immediate challenges remaining when applying post-processing methods. On the one hand, we need to rate the methods when we do not have access to the noiseless matrix. On the other hand, the impact of the methods on the classification accuracy remains to be investigated.

In the design and training of QEKs, one could explore various aspects. A clear question would be the choice of ansatz families. Some key objects of study for this would be the expressivity of different circuits, the dependence on the dataset, the optimal choice of hyperparameters (or, alternatively, how one could perform empirical risk minimization successfully), or how one would build gauge invariant kernel functions [66]. Another major topic is investigating the effect of the barren plateau phenomenon [67–69] in the kernel setting, and subsequently the study of (quantum-aware) cost function alternatives to the target alignment.

Finally, one could explore whether the proposed model can be transferred to more general tasks such as unbalanced binary classification, multi-class classification, or regression.

ACKNOWLEDGMENTS

The authors wish to thank Xanadu for organizing QHack 2021, where the foundations of this work were laid as part of the Open Hackathon Challenge, and the resulting funding. We further want to thank the AWS team for their support and funding that provided us access to the Rigetti and IonQ devices, as well as Sandbox@Alphabet for alpha access to the Floq cloud service, yielding access to the TPU-based quantum simulator. Additionally, we would like to thank Richard Kueng for valuable input on bounds, as well as Jens Eisert and Maria Schuld for valuable feedback. We endorse Scientific CO_2nduct [70] and provide a CO_2 emission table in App. D.

This work was supported by the BMWi under the PlanQK initiative, the BMBF under the RealistiQ initiative, the Cluster of Excellence MATH+ project EF1-7, the European Flagship project PasQuanS and the DFG under Germany's Excellence Strategy Cluster of Excellence Matter and Light for Quantum Computing (ML4Q) EXC2004/1 390534769 and the CRC 183 project B01.

AUTHOR CONTRIBUTIONS

JJM, EGF and PKF built the theory on trainable QEKs. TH and EGF ran the numerics for the trainable QEKs. DW, PKF and JJM built the theory on noise mitigation. DW and PJHSD ran the numerics for noise mitigation. JJM supervised the project. All authors contributed to the discussions and to writing the manuscript.

[1] F. Arute *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[2] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, Variational quantum algorithms, arXiv:2012.09265 (2020).

[3] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, Noisy intermediate-scale quantum (NISQ) algorithms, arXiv:2101.08448 (2021).

[4] O. Vinyals *et al.*, Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature **575**, 350 (2021).

[5] T. B. Brown *et al.*, Language models are few-shot learners, arXiv:2005.14165 (2021).

[6] A. W. Senior *et al.*, Improved protein structure prediction using potentials from deep learning, Nature **577**, 706 (2021).

[7] B. Schlkopf, A. J. Smola, and F. Bach, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (The MIT Press, 2018).

[8] M. Schuld, I. Sinayskiy, and F. Petruccione, An introduction to quantum machine learning, Contemporary Physics **56**, 172 (2015).

[9] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Quantum machine learning, Nature **549**, 195 (2017).

[10] S. Mangini, F. Tacchino, D. Gerace, D. Bajoni, and C. Macchiavello, Quantum computing models for artificial neural networks, arXiv:2102.03879 (2021).

[11] M. Schuld and F. Petruccione, *Supervised learning with quantum computers*, Quantum science and technology (Springer International Publishing, 2018).

[12] P. Wittek, *Quantum machine learning: What quantum computing means to data mining*, Elsevier insights (Elsevier Science, 2014).

[13] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum principal component analysis, Nature Physics **10**, 10.1038/nphys3029 (2014).

[14] M. Schuld, R. Sweke, and J. J. Meyer, Effect of data encoding on the expressive power of variational quantum-machine-learning models, Phys. Rev. A **103**, 032430 (2021).

[15] V. Havlicek, 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 **567**, 10.1038/s41586-019-0980-2 (2019).

[16] M. Schuld and N. Killoran, Quantum Machine Learning in Feature Hilbert Spaces, Phys. Rev. Lett. **122**, 10.1103/PhysRevLett.122.040504 (2019).

[17] T. Kusumoto, K. Mitarai, K. Fujii, M. Kitagawa, and M. Negoro, Experimental quantum kernel machine learning with nuclear spins in a solid, arXiv:1911.12021 (2019).

[18] K. Bartkiewicz, C. Gneiting, A. Cernoch, K. Jiráková, K. Lemr, and F. Nori, Experimental kernel-based quantum machine learning in finite feature space, Scientific Reports **10**, 12356 (2020).

[19] C. Blank, D. K. Park, J.-K. K. Rhee, and F. Petruccione, Quantum classifier with tailored quantum kernel, npj Quantum Information **6**, 1 (2020).

[20] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, Power of data in quantum machine learning, arXiv:2011.01938 (2021).

[21] E. Peters, J. Caldeira, A. Ho, S. Leichenauer, M. Mohseni, H. Neven, P. Spentzouris, D. Strain, and G. N. Perdue, Machine learning of high dimensional data on a noisy quantum processor, arXiv:2101.09581 (2021).

[22] M. Schuld, Quantum machine learning models are kernel methods, arXiv:2101.11020 (2021).

[23] Y. Liu, S. Arunachalam, and K. Temme, A rigorous and robust quantum speed-up in supervised machine learning, arXiv:2010.02174 (2020).

[24] Github code repository, https://github.com/thubregtsen/qhack (2021).

[25] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, Quantum embeddings for machine learning (2020).

[26] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*, Adaptive computation and machine learning (MIT Press, 2002).

[27] B. Schölkopf, A. Smola, and K.-R. Müller, Kernel principal component analysis, in *Artificial Neural Networks — ICANN'97*, Vol. 1327, edited by W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud (Springer Berlin Heidelberg, 1997) pp. 583–588.

[28] C. Saunders, A. Gammerman, and V. Vovk, Ridge regression learning algorithm in dual variables, in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998) p. 515–521.

[29] M. Fanizza, M. Rosati, M. Skotiniotis, J. Calsamiglia, and V. Giovannetti, Beyond the swap test: optimal estimation of quantum state overlap, Phys. Rev. Lett. **124**, 10.1103/PhysRevLett.124.060503 (2020).

[30] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, Quantum fingerprinting, Phys. Rev. Lett. **87**, 167902 (2001).

[31] L. Cincio, Y. Subaşı, A. T. Sornborger, and P. J. Coles, Learning the quantum algorithm for state overlap, New Journal of Physics **20**, 113022 (2018).

[32] H.-Y. Huang, R. Kueng, and J. Preskill, Predicting many properties of a quantum system from very few measurements, Nature Physics **16**, 10.1038/s41567-020-0932-7 (2020).

[33] S. T. Flammia and Y.-K. Liu, Direct Fidelity Estimation from Few Pauli Measurements, Phys. Rev. Lett. **106**, 10.1103 (2011).

[34] T. Wang, D. Zhao, and S. Tian, An overview of kernel alignment and its applications, Artificial Intelligence Review **43**, 179 (2015).

[35] C. Cortes, M. Mohri, and A. Rostamizadeh, Algorithms for learning kernels based on centered alignment, Journal of Machine Learning Research **13**, 795 (2012).

[36] Y. Baram, Learning by kernel polarization, Neural Computation **17**, 1264 (2005).

[37] T. Wang, S. Tian, H. Huang, and D. Deng, Learning by local kernel polarization, Neurocomputing **72**, 3077 (2009).

[38] L. Wang, Feature selection with kernel class separability, IEEE Transactions on Pattern Analysis and Machine Intelligence **30**, 1534 (2008).

[39] Huilin Xiong, M. N. S. Swamy, and M. O. Ahmad, Optimizing the kernel in the empirical feature space, IEEE Transactions on Neural Networks **16**, 460 (2005).

[40] C. H. Nguyen and T. B. Ho, An efficient kernel matrix evaluation measure, Pattern Recognition **41**, 3366 (2008).

[41] N. Cristianini, J. Kandola, A. Elisseeff, and J. Shawe-Taylor, On Kernel Target Alignment, in *Innovations in Machine Learning: Theory and Applications*, Studies in Fuzziness and Soft Computing, edited by D. E. Holmes and L. C. Jain (Springer, Berlin, Heidelberg, 2006).

[42] J. Kandola, J. Shawe-Taylor, and N. Cristianini, *Optimizing Kernel Alignment over Combinations of Kernels*, Tech. Rep. 121 (2002).

[43] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, The theory of variational hybrid quantum-classical algorithms, New Journal of Physics **18**, 023023 (2016).

[44] K. Temme, S. Bravyi, and J. M. Gambetta, Error mitigation for short-depth quantum circuits, Phys. Rev. Lett. **119**, 180509 (2017).

[45] S. Endo, S. C. Benjamin, and Y. Li, Practical quantum error mitigation for near-future applications, Phys. Rev. X **8**, 031027 (2018).

[46] A. Lowe, M. H. Gordon, P. Czarnik, A. Arrasmith, P. J. Coles, and L. Cincio, Unified approach to data-driven quantum error mitigation, arXiv:2011.01157 (2020).

[47] T. Giurgica-Tiron, Y. Hindy, R. LaRose, A. Mari, and W. J. Zeng, Digital zero noise extrapolation for quantum error mitigation, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2020) pp. 306–316.

[48] A. Strikis, D. Qin, Y. Chen, S. C. Benjamin, and Y. Li, Learning-based quantum error mitigation, arXiv:2005.07601 (2021).

[49] B. Koczor, Exponential error suppression for near-term quantum devices, arXiv:2011.05942 (2021).

[50] W. J. Huggins, S. McArdle, T. E. O'Brien, J. Lee, N. C. Rubin, S. Boixo, K. B. Whaley, R. Babbush, and J. R. McClean, Virtual distillation for quantum error mitigation, arXiv:2011.07064 (2021).

[51] R. Latala, Some Estimates of Norms of Random Matrices, Proceedings of the American Mathematical Society **133**, 10.2307/4097777 (2005).

[52] R. Vershynin, Introduction to the non-asymptotic analysis of random matrices, arXiv:1011.3027 (2011).

[53] Z. D. Bai and Y. Q. Yin, Limit of the Smallest Eigenvalue of a Large Dimensional Sample Covariance Matrix, The Annals of Probability **21**, 10.1214/aop/1176989118 (1993).

[54] V. Roth, J. Laub, M. Kawanabe, and J. Buhmann, Optimal cluster preserving embedding of nonmetric proximity data, IEEE Transactions on Pattern Analysis and Machine Intelligence **25**, 1540 (2004).

[55] A. N. Tikhonov, On the stability of inverse problems, Proceedings of the USSR Academy of Sciences **39**, 195 (1943).

[56] T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer, Classification on pairwise proximity data, in *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II* (MIT Press, Cambridge, MA, USA, 1999) p. 438–444.

[57] A. Kandala, K. Temme, A. D. Corcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, Extending the computational reach of a noisy superconducting quantum processor, arXiv:1805.04492 (2018).

[58] W. J. Huggins, S. McArdle, T. E. O'Brien, J. Lee, N. C. Rubin, S. Boixo, K. B. Whaley, R. Babbush, and J. R. McClean, Virtual distillation for quantum error mitigation, arXiv:2011.07064 (2020).

[59] B. Koczor, Exponential error suppression for near-term quantum devices, arXiv:2011.05942 (2020).

[60] X. Wang, Y. Du, Y. Luo, and D. Tao, Towards understanding the power of quantum kernels in the nisq era, arXiv:2103.16774 (2021).

[61] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, Data re-uploading for a universal quantum classifier, Quantum **4**, 10.22331/q-2020-02-06-226 (2020).

[62] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Quantum circuit learning, Phys. Rev. A **98**, 032309 (2018).

[63] K. Wright, K. Beck, S. Debnath, J. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. Pisenti, M. Chmielewski, C. Collins, *et al.*, Benchmarking an 11-qubit quantum computer, Nat. Commun. **10**, 1 (2019).

[64] V. Bergholm *et al.*, Pennylane: Automatic differentiation of hybrid quantum-classical computations., arXiv:1811.04968 (2018).

[65] Amazon braket, https://aws.amazon.com/braket/ (2021).

[66] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, arXiv:2104.13478 (2021).

[67] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, Barren plateaus in quantum neural network training landscapes, Nat. Commun. **9**, 10.1038/s41467-018-07090-4 (2018).

[68] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, Cost function dependent barren plateaus in shallow parametrized quantum circuits, Nat. Commun. **12**, 1791 (2021).

[69] A. Uvarov and J. Biamonte, On barren plateaus and cost function locality in variational quantum algorithms, arXiv:2011.10530 (2020).

[70] Scientific co2nduct, online (2021).

[71] V. Strassen, Gaussian elimination is not optimal, Numerische mathematik **13**, 354 (1969).

[72] Y. T. Lee, A. Sidford, and S. C.-w. Wong, A faster cutting plane method and its implications for combinatorial and convex optimization, in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (IEEE, 2015) pp. 1049–1065.

[73] M. Guţă, J. Kahn, R. Kueng, and J. A. Tropp, Fast state tomography with optimal error bounds, J. Phys. A Math. Theor. **53**, 204001 (2020).

Appendix A: Connection to quantum feature map optimization

Optimizing kernels using the kernel-target alignment as a cost function is closely related to the "metric learning" approach put forward for the training of quantum feature embeddings in Ref. [25].

To understand this approach, we first introduce some notation. We consider a dataset $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}$ that we split in two parts corresponding to the two classes labeled as ± 1. We denote these subsets as $\mathcal{S}_+$ and $\mathcal{S}_-$, respectively. For a given embedding $|\phi_{\boldsymbol{\theta}}(\boldsymbol{x})\rangle$, we can identify both classes with quantum states – we will refer to them as *class states* – simply by

averaging the embedded quantum states

$$\rho_{\pm}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{S}_{\pm}|} \sum_{\boldsymbol{x}\in\mathcal{S}_{\pm}} |\phi_{\boldsymbol{\theta}}(\boldsymbol{x})\rangle\langle\phi_{\boldsymbol{\theta}}(\boldsymbol{x})| \tag{A1}$$

$$= \frac{1}{|\mathcal{S}_{\pm}|} \sum_{\boldsymbol{x}\in\mathcal{S}_{\pm}} \phi_{\boldsymbol{\theta}}(\boldsymbol{x}). \tag{A2}$$

Here, we denoted the density matrix of the embedding as $\phi_{\boldsymbol{\theta}}(\boldsymbol{x}) = |\phi_{\boldsymbol{\theta}}(\boldsymbol{x})\rangle\langle\phi_{\boldsymbol{\theta}}(\boldsymbol{x})|$. The state $\rho_{\pm}$ models an approach where the the encoded datapoint $\boldsymbol{x}$ is uniformly sampled from $\mathcal{S}_{\pm}$.

Ref. [25] suggests to optimize the embedding $|\phi_{\boldsymbol{\theta}}(\boldsymbol{x})\rangle$ by maximizing the Hilbert-Schmidt distance of the class states, i.e.,

$$P(\boldsymbol{\theta}) = \mathrm{Tr}\{(\rho_{+}(\boldsymbol{\theta}) - \rho_{-}(\boldsymbol{\theta}))^2\}. \tag{A3}$$

Its relation to kernel-target alignment becomes apparent if we rewrite the numerator of the kernel-target alignment – the polarity – in terms of these density matrices. We therefore consider the polarity for imbalanced datasets, where we rescale the labels with the number of datapoints in the class. The rescaled labels are denoted as $\hat{y}_j$.

$$\sum_{i,j=1}^{N} \hat{y}_i\hat{y}_j k_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \sum_{i,j=1}^{N} \hat{y}_i\hat{y}_j \langle\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \phi_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\rangle \tag{A4}$$

$$= \left\langle \sum_{i=1} \hat{y}_i\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \sum_{i=1} \hat{y}_i\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \right\rangle \tag{A5}$$

$$= \left\| \sum_{i=1} \hat{y}_i\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \right\|^2. \tag{A6}$$

The polarity is therefore nothing else but the squared norm of $\sum_{i=1} \hat{y}_i\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$, which is a weighted sum of the embedded datapoints. For QEKs, this is equal to the difference of the two class matrices introduced above:

$$\sum_{i=1} \hat{y}_i\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_+\in\mathcal{S}_+} \frac{\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_+)}{|\mathcal{S}_+|} - \sum_{\boldsymbol{x}_-\in\mathcal{S}_-} \frac{\phi_{\boldsymbol{\theta}}(\boldsymbol{x}_-)}{|\mathcal{S}_-|} \tag{A7}$$

$$= \rho_{+}(\boldsymbol{\theta}) - \rho_{-}(\boldsymbol{\theta}). \tag{A8}$$

This means that the polarity is equal to the Hilbert-Schmidt distance introduced in Ref. [25], as found in Eq. (A3)

As already noted in Ref. [25], the polarity can be rewritten as

$$P(\boldsymbol{\theta}) = \mathrm{Tr}\{\rho_{+}(\boldsymbol{\theta})^2 + \rho_{-}(\boldsymbol{\theta})^2 - 2\rho_{+}(\boldsymbol{\theta})\rho_{-}(\boldsymbol{\theta})\}. \tag{A9}$$

Consequently, increasing the polarity translates to an increase in the *purity* of the class states $\mathrm{Tr}\{\rho_{\pm}(\boldsymbol{\theta})\}^2$, thereby encouraging points in the dataset to cluster closer together in feature space. At the same time, this cost function decreases the overlap of the two data embedding states, thereby encouraging them to reside in different corners of the Hilbert space.

However, we are of the opinion that the kernel-target alignment – representing the normalized polarity – is a measure that is easier to interpret and more accessible to numerical optimization than the pure polarity. Ref. [25] proposes a classifier where the overlap of the embedded datapoint with the two class states is computed. The label of the class state with the larger overlap is then assigned to the new datapoint. This corresponds to a kernelized nearest-centroid classification. We conclude that the use of the embedding in a support vector machine allows for more sophisticated decision boundaries than the method proposed in Ref. [25].

Appendix B: Details on post-processing methods

1. Runtimes and output properties

The post-processing methods we introduced in Sec. V D and V B differ in their classical and quantum computational cost and in the properties of the output matrix.

The regularization methods R-TIK and R-THR require the computation of the smallest eigenvalue and of the full eigenvalue decomposition respectively, which has classical complexity $\mathcal{O}(n^3)$ with naive methods but more realistically scales like matrix multiplication for relevant sizes with $\mathcal{O}(n^{2.8})$ (Strassen algorithm [71])[5]. The worst case scaling for R-SDP is $\mathcal{O}(n^{3.8})$, again assuming the Strassen algorithm for matrix multiplication and considering that we use n constraints to fix the diagonal entries [71, 72]. In our experiments on datasets with 60 datapoints, the former two methods had negligible computational cost, whereas R-SDP took $0.5s$ on average for this rather small matrix. In addition to this large difference in the prefactor, some additional tests for random matrices confirmed a significantly worse scaling of the cost for R-SDP compared to R-TIK and R-THR.

As they only act on the kernels spectrum, R-TIK and R-THR preserve the eigenbasis of the kernel matrix, a potentially relevant property for the classification task. On the contrary, R-SDP does not preserve the eigenbasis but ensures that the output kernel matrix has the correct diagonal entries.

For the proposed mitigation methods, additional quantum computation is required in order to determine the diagonal entries, which in turn are used to estimate the depolarizing survival probabilities. The number of required entries is 1, $n_{\mathrm{mean}} \in [1, n]$ and n for M-SINGLE, M-MEAN and M-SPLIT, respectively, which then should be measured as often as the other matrix entries. While estimating the probabilities has negligible cost, the modification of the matrix requires $\mathcal{O}(n^2)$ classical computation resources[6].

Considering Eq. (38), we see that our mitigation methods estimate the survival probability λ_i to be larger than one for $K_{ii}^{(\mathrm{dev})} > 1$ and to be imaginary if $K_{ii}^{(\mathrm{dev})} < 2^{-N}$, both being

[5] If this was to be a bottle neck, the full matrix multiplication may be skipped when multiplying the kernel matrix with vectors only.

[6] This may again be improved if we are not interested in the fully computed matrix but e.g. in multiplying it with vectors, should it ever become a major resource requirement.

unreasonable estimates. The first will only ever occur if a previous post-processing method increased the diagonal element $K_{ii}^{(\mathrm{dev})}$ too far, as a QPU itself will not output measurement probabilities above 1. The second may occur in the presence of very strong noise that suppresses the exact value of 1 to 2^{-N}, which would presumably imply the QPU output to be impracticably flawed anyways. For M-SINGLE (M-MEAN), the same reasoning holds for the single measured entry (for the average of the considered diagonal entries), i.e. in particular for M-MEAN we are unlikely to run into either of the above problems.

Even if the estimated survival probabilities λ_i lie in the physically meaningful range $[0, 1]$, the mitigation might still produce kernel matrix entries that are not valid probabilities and thus can impossibly be the result of a real QEK evaluation. For a given noisy matrix entry $K_{ij}^{(\mathrm{dev})}$, this happens if

$$K_{ij}^{(\mathrm{dev})} \notin [\varepsilon, \lambda_i\lambda_j + \varepsilon], \tag{B1}$$

where we abbreviated $\varepsilon = 2^{-N}(1 - \lambda_i\lambda_j)$ and the estimated probabilities fulfill $\lambda_i = \lambda_j$ for M-SINGLE and M-MEAN. Note that $\lambda_i \approx 1$ and $\varepsilon \ll 1$ for reasonable survival rates.

In conclusion, even though there are extreme cases in which our methods might transform the noisy matrix into an invalid kernel matrix, we do not expect these problems to play any role because such extreme noise levels likely would render the QPU output useless.

Note that the error bound in operator distance derived in Sec. V C is valid for the deviation of the statistical estimator from the noiseless kernel matrix K or from the device-noisy kernel matrix $K^{(\mathrm{dev})}$. When applying post-processing methods however, this bound may not transfer to the output $K^{(\mathrm{post})}$ in general. Consequently, while being designed to counter device and finite sampling noise, the analytic error bound might become worse.

For R-THR however, this bound is provably maintained [73]: Splitting the indefinite matrix $\bar{K}_M$ into the difference of two positive semi-definite matrices K_+ and K_- with disjoint support, identifying $\mathsf{R\text{-}THR}(\bar{K}_M) = K_+$ and calculating the distance between the approximand[7] and $\bar{K}_M$ yields

$$\begin{aligned}
\bar{K}_M =:& K_+ - K_- && \text{(B2)}\\
\|K - \bar{K}_M\|_\infty =& \|K - K_+ + K_-\|_\infty && \text{(B3)}\\
=& \max_{\|x\|_2=1} [x^T(K - K_+)^2 x && \text{(B4)}\\
&\qquad + \underbrace{x^T(KK_- + K_-K + K_-^2)x}_{\geq 0}]\\
\geq& \max_{\|x\|_2=1} x^T(K - K_+)^2 x && \text{(B5)}\\
=& \|K - K_+\|_\infty && \text{(B6)}\\
=& \|K - \mathsf{R\text{-}THR}(\bar{K}_M)\|_\infty && \text{(B7)}
\end{aligned}$$

where we used the positive semi-definiteness of K_- and that $K_\pm K_\mp = 0$ due to the disjoint support.

[7] We here show the calculation when approximating K. It has to be replaced by $K^{(\mathrm{dev})}$ accordingly when approximating the device-noisy matrix by sampling.

2. Comparison of post-processing strategies

There are many combinations of the post-processing techniques to choose from in order to counter both device noise and finite sampling noise.

First, we apply a regularization R_1, including the option to not modify $K^{(\mathrm{dev})}$ at all (Id). Second, we perform device noise mitigation M and third, we regularize again with R_2.

For the two regularization steps $\mathsf{R}_{1,2}$, we may apply Tikhonov regularization (R-TIK), thresholding (R-THR) or the *semi-definite program* (SDP) fixing the diagonal while thresholding (R-SDP), see Sec. V D. For the mitigation step, we choose from single survival probability estimation based on a single (M-SINGLE) or the mean (M-MEAN) diagonal entry of $\bar{K}_M$, and survival probability estimation per feature embedding (M-SPLIT), see Sec. V B.

Naively, this yields 64 combinations when including the trivial transformation Id, out of which some are identical, e.g. $\mathrm{Id}, \mathrm{Id}, \mathsf{R}$ and $\mathsf{R}, \mathrm{Id}, \mathrm{Id}$. In addition, there are special combinations in which methods effectively act like Id: First, combinations of the form $\mathsf{R\text{-}SDP}, \mathsf{M}, \mathsf{R}_2$ for which M already receives a positive semi-definite input matrix with correct diagonal entries and thus will estimate the survival probability to be 1. Second, some combinations without mitigation (namely $\mathsf{R\text{-}TIK}/\mathsf{R\text{-}THR}, \mathrm{Id}, \mathsf{R\text{-}TIK}/\mathsf{R\text{-}THR}$) in which R_2 would be redundant. Here we already excluded the combinations obeying the first pattern. Excluding duplicates and these "reducible" combinations, we obtain 42 reasonable, distinct strategies (including $\mathrm{Id}, \mathrm{Id}, \mathrm{Id}$) and for each of the outcomes $K^{(\mathrm{post})}$ we compute the kernel alignment (see Eq. 26) with the noiseless matrix K.

In Fig. 10 we only showed the best out of the resulting 42 combinations, rated by the lowest alignment across all base survival probabilities and numbers of circuit evaluations. This best combination is a single application of R-THR (see Eq. (48)), not making use of any mitigation or the second regularization step. However, we note that the influence of sampling noise is rather large in the chosen domains of M and λ_0, such that the rating by lowest achieved accuracy favors methods that are designed to counter sampling noise, such as R-THR.

Choosing the strategy in this way, our post-processing increases the alignment significantly (by up to 43.5%, in the regime of small M) and systematically (only negligible deterioration for $M \to \infty$), allowing for an improved estimate of the kernel matrix with fewer circuit evaluations.

In Fig. 12 we evaluate the combinations of regularization and mitigation in more detail and it becomes apparent that the best choice depends on the noise regime. We immediately see that in the domain of high noise (small numbers of circuit evaluations and lower survival probability, lower right) the result of our simple ranking in Sec. VII B is confirmed and thresholding consistently is the best post-processing method (combination 3).

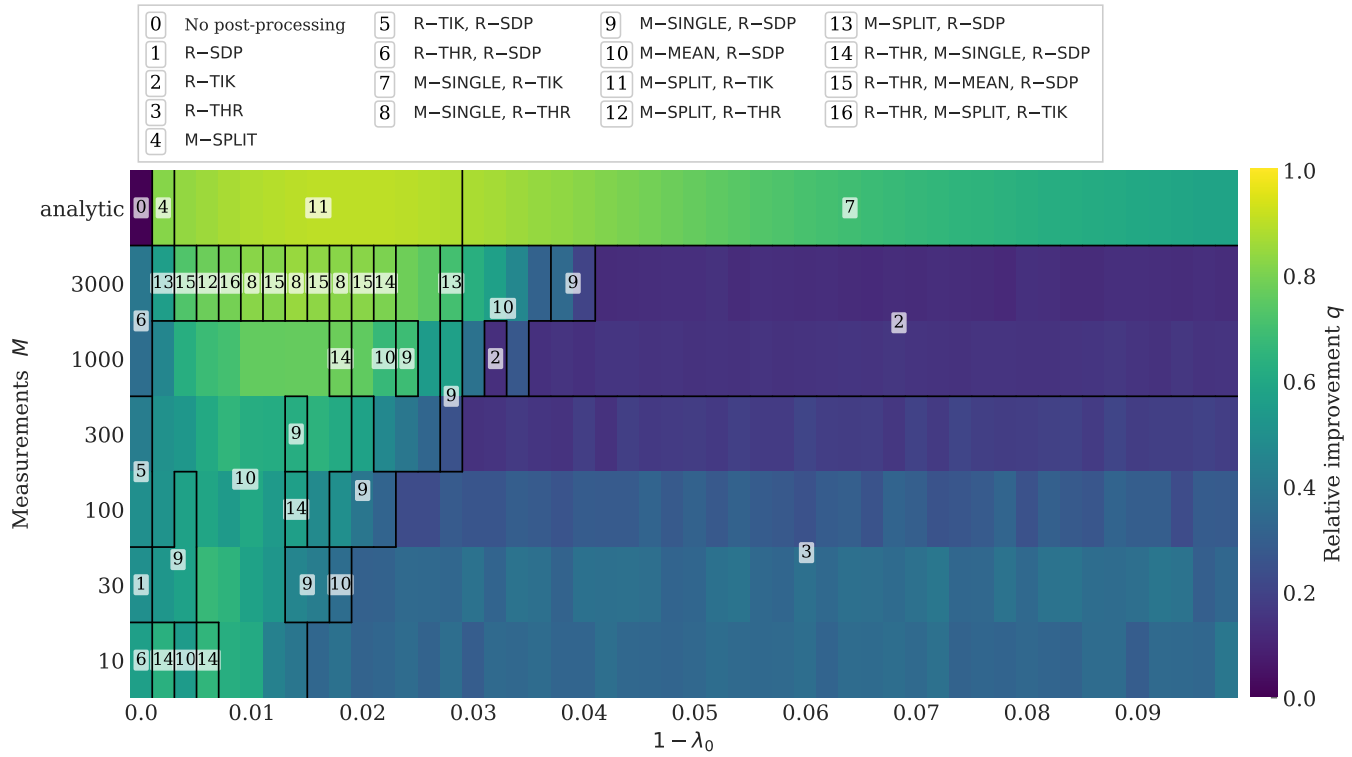

Figure 12. Relative improvement q in alignment (see Eq. (52)) between noisy and noiseless kernel matrices as in Fig. 10 but for the best combination of post-processing steps *per point*. We vary the base survival probability λ_0 of the depolarizing noise model (see Sec. C 5) and the number of circuit evaluations M per kernel matrix entry, including the analytically exact limit $M \to \infty$ in the first row. For details on the noise mitigation and regularization methods see Sec. V B and V D, respectively. Black solid lines separate areas within which the same combination is best, the numbers label the combinations.

When increasing M while at the lower device noise level, Tikhonov regularization becomes more favorable than thresholding and still no device noise mitigation method is able to improve the matrix further (combination 2). This is remarkable because our combinations would allow for multiple processing steps to counter both, sampling and device noise.

For higher λ_0 we observe high variation in which combination is best, because many of them yield very similar alignments with the exact matrix so that statistical fluctuations become relevant. However, for the majority of these datapoints with higher survival probabilities, the combination of single probability estimation based on a single or the mean of the matrix diagonal (M-SINGLE/M-MEAN) followed by SDP-based regularization (R-SDP) is best (combinations 9 and 10).

Finally, for matrices without sampling noise our device noise mitigation techniques combined with Tikhonov regularization are the best choice, delivering far better results than simple regularization (combinations 5 and 8). This indeed confirms that the high-level noise model of global depolarization, on which the mitigation techniques are based, is able to grasp some of the essential influence the device noise (of our more realistic depolarizing noise model) has. Here the single probability estimation M-SINGLE seems to be better suited for lower survival probabilities and M-SPLIT for higher λ_0.

The complex appearance of Fig. 12 underlines that the best choice of post-processing depends significantly on the noise level in the device and – in relation to this level – how many circuit evaluations are used for the kernel evaluation, so one should choose the method carefully. The benefit of doing so compared to the first, simple rating in Sec. VII B is an increase of the best achieved alignment improvement q from 43.5% to 84.6% (or even to 90.2% when considering the results without sampling noise in the first row of Fig. 12).

Finally, we note that the ranking of the post-processing combinations based on the alignment of $K^{(\mathrm{post})}$ with the noiseless kernel matrix K is not possible in applications that truely require the QPU, as K is not available in this case. Instead one could evaluate the restored matrices based on their alignment with the ideal target matrix K^*. We compared the ranking resulting from this idea with the one presented above and did not find any systematic relations between them, so that this application-oriented surrogate method to reconstruct K may be discarded. Whether it provides a better kernel matrix for classification is an open question, but if K^* was optimal, there would not be any use in computing K from the start.

We conclude that choosing the post-processing in practice remains challenging and requires a systematic analysis of both finite sampling and device noise effects on larger-scale kernel matrices.

Appendix C: Additional information on numerical experiments

1. Checkerboard

The checkerboard dataset is used for testing our error mitigation techniques for medium size kernels on a classical simulator, as well as for the noiseless simulation experiment. The dataset consists of 30 training and 30 test datapoints, and was generated as follows. In the domain $[0,1]^2$ get a 4×4 grid with sites i, j at coordinates $((2i+1)/8, (2j+1)/8)$ to prevent overlap between centroids and spilling out of the fixed domain. Next, we sampled points uniformly centered about each grid site. At the end, we assigned alternating classes to each of the sites, and finished by assigning all swarms of points the class corresponding to their centroid.

2. Symmetric donuts

The symmetric donuts dataset ought to set the grounds for running a smaller kernel on actual quantum hardware. This artificial dataset has the same size as the previous one: 60 training and 60 test datapoints. The datapoints are generated by sampling points uniformly at random from a circle of radius $\sqrt{2}/2$ and then labeling them according to whether they fall within an inner circle of radius $1/2$ or without. We do this one time centering the circles on the x-axis, on the point $(1,0)$, giving the inner points label 1 and the outer ones label -1. Next, we repeat the process for circles centered about the point $(-1,0)$ and this time exchange the labels: the inner point class is now -1 and the outer $+1$. This way we obtain a dataset contained in the domain $[-(3+\sqrt{2})/2, (3+\sqrt{2})/2] \times [-\sqrt{2}/2, \sqrt{2}/2]$.

3. Semi-artificial MNIST

Moving our experiment closer to real-world data, we have also considered MNIST images. This dataset contains handwritten digits in images that have dimensions of 28x28 and a gray-scale from zero to 255, which we normalize to the range from zero to one. To construct a semi-artificial dataset, we have randomly chosen 500 images with labels of zero and one each, sampled three pixels with gray-scale values larger than 0.95 from each, and added all these coordinates to construct "zero-base" and "one-base" sets. We then constructed the dataset "zero" by selecting all coordinates in "zero-base" that are not contained in "one-base" and analogously for the dataset "one". The dataset "non-zero" is a copy of dataset "one-base", "non-one" is a copy of dataset "zero-base". The datasets are shown in the top six plots of Fig. 13.

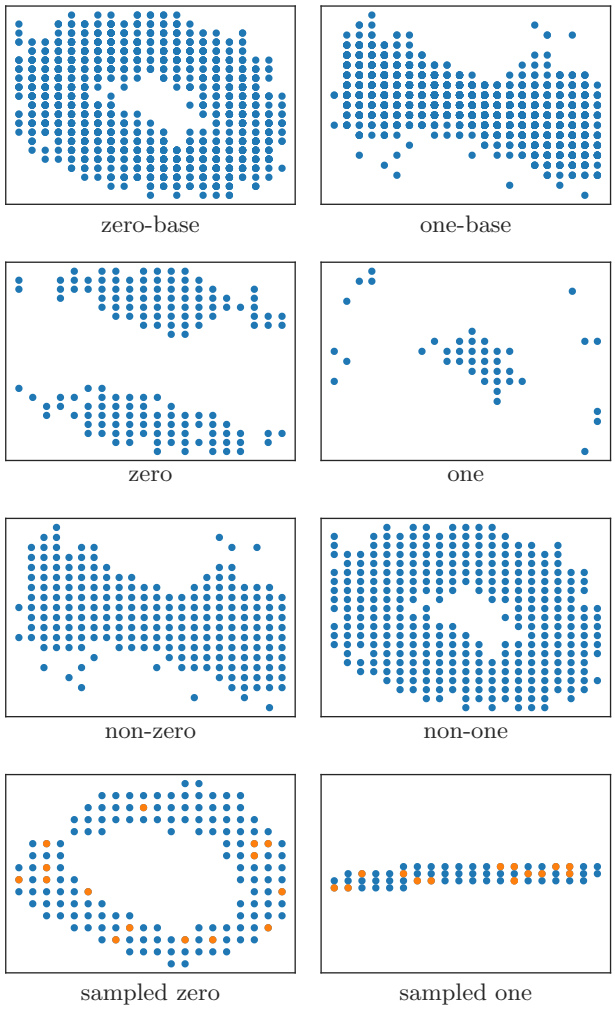

Figure 13. Various subsets of the MNIST dataset. All images appear rotated 90 degrees counter-clockwise.

4. Ensemble MNIST

We previously used the semi-artificial MNIST dataset to train a classifier for classifying single pixel coordinates from a combined pool of 500 images belonging to an image labeled zero (one) or not. We now use several of these classifiers in an ensemble, to classify "zero vs one" for individual images. This classification is done via the following steps: First, we sample 15 pixels with a gray-scale value larger than 0.95 from our image, and store them as coordinates. This is shown in the bottom two plots of Fig. 13. We then use the trained kernel classifiers to classify "zero" vs. "not zero" and "one" vs. "not one". Finally, we perform a majority vote for each coordinate. If the relative vote wins by less than two votes, we re-run our method with newly sampled points.

5. Simulating device noise by depolarization

For the simulation of device noise in Sec. VII B we use the following noise model: After each unitary gate we apply single-qubit depolarizing noise channels $\mathcal{D}_\lambda$ to each qubit the gate acted on (see Eq. (29) with $N = 1$).

Recalling the discussion in Sec. V A, we remark that the qubit-wise depolarizing channel does commute with single-qubit but not with multi-qubit gates like the ring of controlled gates in our embedding circuit. In this sense our model properly captures the case in which the device noise invalidates the adjoint approach, potentially destroying the positive semi-definiteness of the kernel matrix, and our post-processing strategies are challenged to correct this deviation.

The *base survival probability* λ_0 quantifies the overall noise strength. However, it is reasonable to expect that the noise strength for a specific gate on a QPU depends on the duration of the pulses that implement the gate, leading to different effective noise levels for different embedded datapoints. In order to capture this dependence, we rescale the base survival probability λ_0 for a rotation gate about the angle θ according to

$$\lambda = \left(1 - \frac{\theta}{2\pi}\right) + \lambda_0 \frac{\theta}{2\pi} \tag{C1}$$

and fix the survival probability of the Hadamard and idling gate to $(1 + \lambda_0)/2$ and $(1 + 3\lambda_0)/4$, respectively.

We do not simulate any device readout error explicitly but assume the presented implementation of depolarizing noise to represent the full device noise closely enough. This assumption seems to be valid considering our results in Secs. VII B and VII C and the accordance between them.

Appendix D: CO_2 emission table

Numerical simulations	
Total Kernel Hours [h]	7250
Thermal Design Power Per Kernel [W]	4.6
Total Energy Consumption Simulations [kWh]	32.8
Average Emission Of CO_2 In Germany/USA [kg/kWh]	0.47
Total CO_2-Emission For Numerical Simulations [kg]	15.5
Estimated CO_2-Emission For QPU usage [kg]	21.4
Were The Emissions Offset?	**Yes**
Total CO_2-Emission [kg]	36.9