BNL-225479-2024-JAAM

Parallel hybrid quantum-classical machine learning for kernelized time-series classification

J. Baker, K. Yu

To be published in "Quantum Machine Intelligence"

March 2024

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Advanced Scientific Computing Research (ASCR)

DISCLAIMER

Quantum Machine Intelligence _####################_
https://doi.org/10.1007/s42484-024-00149-0

RESEARCH ARTICLE

Parallel hybrid quantum-classical machine learning for kernelized time-series classification

Jack S. Baker[1] · Gilchan Park[2] · Kwangmin Yu[2] · Ara Ghukasyan[1] · Oktay Goktas[1] · Santosh Kumar Radha[1]

Received: 30 June 2023 / Accepted: 6 February 2024

Abstract
Supervised time-series classification garners widespread interest because of its applicability throughout a broad application domain including finance, astronomy, biosensors, and many others. In this work, we tackle this problem with hybrid quantum-classical machine learning, deducing pairwise temporal relationships between time-series instances using a time-series Hamiltonian kernel (TSHK). A TSHK is constructed with a sum of inner products generated by quantum states evolved using a parameterized time evolution operator. This sum is then optimally weighted using techniques derived from multiple kernel learning. Because we treat the kernel weighting step as a differentiable convex optimization problem, our method can be regarded as an end-to-end learnable hybrid quantum-classical-convex neural network, or QCC-net, whose output is a data set-generalized kernel function suitable for use in any kernelized machine learning technique such as the support vector machine (SVM). Using our TSHK as input to a SVM, we classify univariate and multivariate time-series using quantum circuit simulators and demonstrate the efficient parallel deployment of the algorithm to 127-qubit superconducting quantum processors using quantum multi-programming.

Keywords Quantum machine learning · Time-series · Kernel methods · Quantum multi-programming · Convex optimization

1 Introduction

Processes and systems which produce observable characteristics evolving with time are present in topics as disparate as finance, sensor technologies, medicine, astronomy, and many others. As a result, new techniques for time-series analysis have become among the most sought after in machine learning (ML) where we seek to learn temporal trends and correlations from time-series data to perform classification (Fawaz et al. 2019), anomaly detection (Blázquez-García et al. 2021; Choi et al. 2021), regression (Clark et al. 2020), forecasting (Deb et al. 2017; Torres et al. 2021), and to generate synthetic time-series instances (Zhang et al. 2018). Focusing on classification, classical ML has provided a zoo of algorithms where the present state-of-the-art is centered around deep learning. Popular approaches include recurrent neural networks like long-short-term-memory (Hochreiter Schmidhuber 1997) (and its variations (Melis et al. 2019; Nguyen et al. 2020)), gated recurrent units and, more recently, transformer networks (Vaswani et al. 2017; Yang et al. 2021; Zerveas et al. 2021).

While popular for static data (i.e., data without time dependence), kernelized methods like the well-known support vector machine (SVM) (Cortes Vapnik 1995) see limited use in time-series analysis. Although time-series kernel methods have been proposed (Badiane et al. 2018; Bailly 2018; Fábregues de los Santos 2017; Rüping 2001) and used as analytical tools for interpreting deep learning models (Tino 2020), none are tailored to encode temporal trends by way of an explicit and learnable time-dependent inner product space. Within classical ML, it is not clear how such a space can be constructed in a non-trivial manner. In this work, we show that time evolution as generated by a parameterized Hamiltonian operator within quantum mechanics is a natural approach for achieving such an objective. Indeed, by combining inner products evolved to different points of time, we achieve a quantum kernel function adapted for time-

✉ Kwangmin Yu
kyu@bnl.gov

Santosh Kumar Radha
research@agnostiq.ai

[1] Agnostiq Inc., 325 Front St W, Toronto, ON M5V 2Y1, Canada

[2] Computational Science Initiative, Brookhaven National Laboratory, Upton, New York, NY 11973, USA

series data which we call the time-series Hamiltonian kernel (TSHK). Furthermore, we discuss situations where training a TSHK is useful and provide theoretical tools to analyze the time dependence of the learnt inner product spaces.

The construction of the TSHK is tied to a field known as multiple kernel learning (MKL). In MKL, multiple kernel functions are combined using various tactics (Gönen Alpaydın 2011a; Aiolli Donini 2015) to give rise to a combined kernel that is by some measure superior. The TSHK is a combined kernel built using a weighted linear combination of quantum kernel functions each defined at different instances of time t. These weights are chosen to maximize the separation between labeled data classes as was proposed in the well-known EasyMKL (Aiolli Donini 2015) algorithm. Furthermore, as was originally suggested in Ghukasyan et al. (2023), the variational parameters of quantum kernels and the kernel weights can be obtained simultaneously by training an end-to-end learnable Quantum-Classical-Convex neural network (QCC-net). The network producing the TSHK can then be regarded as a temporally aware variation of the original QCC-net.

Currently in the noisy intermediate-scale quantum (NISQ) era (Bharti et al. 2022), various sources of noise prevent the implementation of deep quantum circuits on real hardware. In light of this, we must ensure that our quantum circuits implementing the TSHK are sufficiently shallow. Conventionally, however, the required time evolution operators are implemented using a potentially high-depth Lie-Suzuki-Trotter expansion (Wiebe et al. 2010). Fortunately, built upon the eigendecomposition approach originally used in the *variational fast forwarding* algorithm (Cîrstoiu et al. 2020), a family of time evolution operators can be implemented with low depth variational circuits. The space of accessible time evolution operators can be controlled by modifying the depth and structure of the circuit Ansätz implementing the eigendecomposition. Several works (Radha 2021; Horowitz et al. 2022; Baker et al. 2022; Gibbs et al. 2022; Caro et al. 2022) have now appeared leveraging this approach to learn constraint-enforcing mixing operators in the Quantum Approximate Optimization Algorithm (Radha 2021), generate synthetic time-series instances (Horowitz et al. 2022), and detect anomalous behavior in time-series data (Baker et al. 2022). The generalization bounds of this approach have also been studied (Gibbs et al. 2022; Caro et al. 2022).

Although the eigendecomposition approach does allow for a shallow circuit implementation of the TSHK, the number of computations required to compute the kernel matrices scales with the square of the size of the training data set and linearly with the length of the time series. Fortunately, the computations of kernel matrix elements are independent of one another allowing them to be computed in parallel. Indeed, a new approach to efficiently utilize contemporary NISQ computers is by overlapping multiple quantum circuits (Das et al. 2019). This method, called Quantum Multi-Programming (QMP), utilizes NISQ devices by executing multiple quantum circuits concurrently. In this work, we use QMP to compute the TSHK in parallel. Compared to serial execution, we achieve a notable speed-up (at least 35 times) with QMP without loss of accuracy using two 127-qubit IBM quantum computers. In light of this large speed-up, this work demonstrates the practical utility of parallelism in hybrid quantum-classical ML workflows.

The rest of this work is organized as follows: in Section 2, we describe how we achieve a time-dependent inner product space and show how a weighted sum of inner products can be used to construct the TSHK (Section 2.1). Also within this section, we show how the TSHK can be trained using a QCC-net (Section 2.2) and show how the resulting TSHK can be plugged in to a SVM to perform time-series classifications (Section 2.3). In Section 3, we describe in detail what is meant by time-dependence in the trained kernel function and show how one can efficiently probe this property. In Section 4, we demonstrate SVM classification using the TSHK for a synthetic multivariate time-series (Section 4.1) and for a real univariate time-series: the well-known gun-point data set (Ratanamahatana Keogh 2005) (Section 4.2). Our experiments using QMP on 127 qubit superconducting transmon chips are contained within Section 5. The QMP approach is described in detail in Section 5.1, and pertinent QMP design considerations are explained in Section 5.2. The results of our QMP experiments are shown and discussed in Section 5.3. Finally, in Section 6, we conclude and remark on future directions in hybrid quantum-classical ML exploiting parallelism.

2 Algorithm description

2.1 Time-dependent inner product space

We begin with a formal definition of the type of data we are working with: time-series. A time-series $\boldsymbol{x}$ is a sequence of $p \in \mathbb{Z}^+$ observations from a process/system arranged in chronological order

$$\boldsymbol{x} := (\boldsymbol{x}_t : t \in T), \quad T := (t_l : 1 \leq l \leq p) \tag{1}$$

where $\boldsymbol{x}_t = [{}^1x_t, {}^2x_t, \ldots {}^dx_t] \in \mathbb{R}^d$, $d \in \mathbb{Z}^+$, $l \in \mathbb{Z}^+$ and time $t \in \mathbb{R}^+$. In what follows, we devise a technique for obtaining a trainable kernel function adapted for time-series data of the form defined above by (i) formulating a time-dependent quantum inner product space and (ii) using methods from the classical MKL literature (Aiolli Donini 2015; Gönen Alpaydın 2011b) to find an optimally weighted sum of inner products from different points in time.

The first step towards achieving our goal is to inject time dependence into a quantum state. We achieve this through operating on the n-qubit state of zeroes $|0\rangle^{\otimes n}$ using a parameterized unitary time evolution operator $e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'}$ where $t' \in \mathbb{R}^+$ followed by a parameterized unitary matrix $U(\boldsymbol{x}_t, \boldsymbol{\alpha})$ used to embed $\boldsymbol{x}_t$ within the quantum state

$$|\boldsymbol{x}_t, t', \boldsymbol{\theta}\rangle := U(\boldsymbol{x}_t, \boldsymbol{\alpha}) e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'} |0\rangle^{\otimes n} \tag{2}$$

for real-valued set of free parameter vectors $\boldsymbol{\theta} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$ where the embedding unitary matrix $U(\boldsymbol{x}_t, \boldsymbol{\alpha})$ is permitted to be any quantum feature map (Schuld Killoran 2019; Havlíček et al. 2019) implemented using a layered parameterized quantum circuit. It should be stressed that $U(\boldsymbol{x}_t, \boldsymbol{\alpha})$ is a static map. That is, despite the subscript t bared by the argument $\boldsymbol{x}_t$, the resulting unitary matrix is time-independent. Explicitly, should we have $\boldsymbol{x}_{t_1} = \boldsymbol{x}_{t_2}$, $t_1 \neq t_2$ the unitary matrices $U(\boldsymbol{x}_{t_1}, \boldsymbol{\alpha})$ and $U(\boldsymbol{x}_{t_2}, \boldsymbol{\alpha})$ are indistinguishable. Time-dependence only comes as a result of the time evolution operator $e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'}$ which we write as an eigendecomposition

$$V_{t'}(\boldsymbol{\beta}, \boldsymbol{\gamma}) := W^\dagger(\boldsymbol{\beta}) D(\boldsymbol{\gamma}, t') W(\boldsymbol{\beta}) = e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'} \tag{3}$$

for parameterized unitary matrix of eigenvectors $W(\boldsymbol{\beta})$ and parameterized time-encoded diagonal unitary matrix $D(\boldsymbol{\gamma}, t')$. The above equality with $e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'}$ holds following Stone's theorem for strongly continuous one-parameter unitary groups (Stone 1932). The strength of this approach lies in implementing each unitary matrix in the eigendecomposition as a layered parameterized quantum circuit allowing one to control the search space of unitary time evolution operators. Should a small number of layers prove sufficient to optimize the forthcoming loss function (Eq. 8), our algorithm becomes suitable for present generation NISQ computers. For notational short-hand, it is convenient to write a combined unitary matrix as a product of the static and time-dependent parts

$$\Omega(\boldsymbol{x}_t, t', \boldsymbol{\theta}) := U(\boldsymbol{x}_t, \boldsymbol{\alpha}) V_{t'}(\boldsymbol{\beta}, \boldsymbol{\gamma}). \tag{4}$$

We can now begin to formally derive a time-dependent inner product. To do so, we first must regard the density matrix $\rho_{t'}(\boldsymbol{x}_t, \boldsymbol{\theta}) = |\boldsymbol{x}_t, t', \boldsymbol{\theta}\rangle\langle \boldsymbol{x}_t, t', \boldsymbol{\theta}|$ as the unambiguous feature vector embedding the classical data point $\boldsymbol{x}_t$ (regarding $|\boldsymbol{x}_t, t', \boldsymbol{\theta}\rangle$ as the feature vector is ambiguous as the space $\mathbb{C}^{2^n}$ is only physically defined up to a global phase (Havlíček et al. 2019)). We now define a kernel by setting $t' \to t$ and taking the Frobenius inner product $\langle \rho_t(\boldsymbol{x}'_t, \boldsymbol{\theta}), \rho_t(\boldsymbol{x}_t, \boldsymbol{\theta})\rangle$. Equivalently,

$$\begin{aligned}\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta}) &:= \mathrm{Tr}[\rho_t(\boldsymbol{x}'_t, \boldsymbol{\theta}) \rho_t(\boldsymbol{x}_t, \boldsymbol{\theta})] \\ &= |\langle \boldsymbol{x}'_t, t, \boldsymbol{\theta} | \boldsymbol{x}_t, t, \boldsymbol{\theta}\rangle|^2 \end{aligned} \tag{5}$$

where $\boldsymbol{x}'_t$ is an element of another time-series instance $\boldsymbol{x}'$ defined analogously to Eq. 1. Equation 5 can be estimated practically by preparing the state $\Omega^\dagger(\boldsymbol{x}'_t, t, \boldsymbol{\theta}) \Omega(\boldsymbol{x}_t, t, \boldsymbol{\theta}) |0\rangle^{\otimes n}$ and taking the expectation value of the projector on the state of zeros $P_0 = |0\rangle^{\otimes n}\langle 0|^{\otimes n}$. This requires repeated preparation and measurement of $\Omega^\dagger(\boldsymbol{x}_t, t, \boldsymbol{\theta}) \Omega(\boldsymbol{x}_t, t, \boldsymbol{\theta}) |0\rangle^{\otimes n}$ in the computational basis to estimate the probability of measuring the n-bit string of zeros $00\ldots0$.

Equation 5 fulfils the symmetry requirement of a valid kernel function. That is, we have $\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta}) = \kappa_t(\boldsymbol{x}'_t, \boldsymbol{x}_t, \boldsymbol{\theta})$ for any choice of the two arguments $\boldsymbol{x}_t, \boldsymbol{x}'_t \in \mathbb{R}$ or the time argument t. Indeed, the achievement of this symmetry motivates the order in which unitary operations are applied in Eq. 2. If we were to switch the order in which we apply of $U(\boldsymbol{x}_t, \boldsymbol{\alpha})$ and $e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'}$ to $|0\rangle^{\otimes n}$, we yield the state $|\boldsymbol{x}_t, t', \boldsymbol{\theta}\rangle_G := e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t'} U(\boldsymbol{x}_t, \boldsymbol{\alpha}) |0\rangle^{\otimes n}$. Computing the inner product of this state at $t' = t$ with another state at $t' = t'$ and multiplying the result by its own complex conjugate, we yield the quantity $\mathbb{G} := |_G\langle x_t, \boldsymbol{\alpha}| e^{-iH(t-t')} |x'_t, \boldsymbol{\alpha}\rangle_G|^2$ where we have taken $|x_t, \boldsymbol{\alpha}\rangle_G := U(\boldsymbol{x}_t, \boldsymbol{\alpha}) |0\rangle^{\otimes n}$. We see immediately that for $t \neq t'$, this expression is *not* a valid kernel as the symmetry requirement detailed above is violated. It is only at $t = t'$ that symmetry is recovered but at the expense of losing time dependence entirely as the time evolution operator becomes the identity. It is worth pointing out that while a time-dependent kernel function is not achievable in this way, the result is otherwise important. That is, interestingly, $\mathbb{G}$ is equivalent to the product of the Green's function propagator (hence the subscript G in the above) from quantum field theory (Bjorken Drell 1965) with its complex conjugate $G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t; \boldsymbol{x}'_t, t') G^\star_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t; \boldsymbol{x}'_t, t')$. Using parameterized Green's function propagators could prove an interesting avenue for non-kernelized time-series analysis in QML but is beyond the scope of this work.

Returning now to the kernel function defined in Eq. 5, we compute it $\forall t \in T$ and combine the p-terms in a weighted sum to achieve a combined kernel function measuring the similarity between a pair of time-series instances $(\boldsymbol{x}, \boldsymbol{x}')$

$$\kappa(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{\theta}, \boldsymbol{\eta}) := \sum_{t \in T} \eta_t \cdot \kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta}) \tag{6}$$

for kernel coefficient vector $\boldsymbol{\eta} := [\eta_t | t \in T] \in \mathbb{R}^p$ subject to the constraint $\sum_{t \in T} \eta_t = 1$. We call Eq. (6) the TSHK. The motivation for such a form of kernel function is to allow the measured similarity between a pair of time-series $(\boldsymbol{x}, \boldsymbol{x}')$ to be influenced by differently weighted contributions at distinct points in time. Should the resulting kernel be used in a

classification algorithm like SVM, this allows for "smoking gun" values of t to discriminate between different classes of time-series. In the next section, we present a strategy for optimally setting the kernel parameters $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ where the exact sense of optimality is to be defined.

Before we advance to the next section, however, now is a good time to discuss what structures of time series data may be learnt well by the kernel function of Eq. 6 and for which we may expect performance to be no better than time independent kernels. It is clear from the form of Eq. 6 and the time-dependent kernels which construct it (Eq. 5) that great emphasis is placed on the notion of a "shared time axis," a significant inductive bias of our approach. That is, we expect a learnable TSHK when each instance of the training time series data begins at a common initial condition in time. Examples of such data include the temperature in different locations for each day of the year (where we may classify the southern hemisphere versus the northern hemisphere) or traffic (the number of detected vehicles) on roadways given at some frequency on different days (where we may classify weekday versus weekend traffic). In the absence of such a shared time axis, we may not expect a learnable TSHK. For

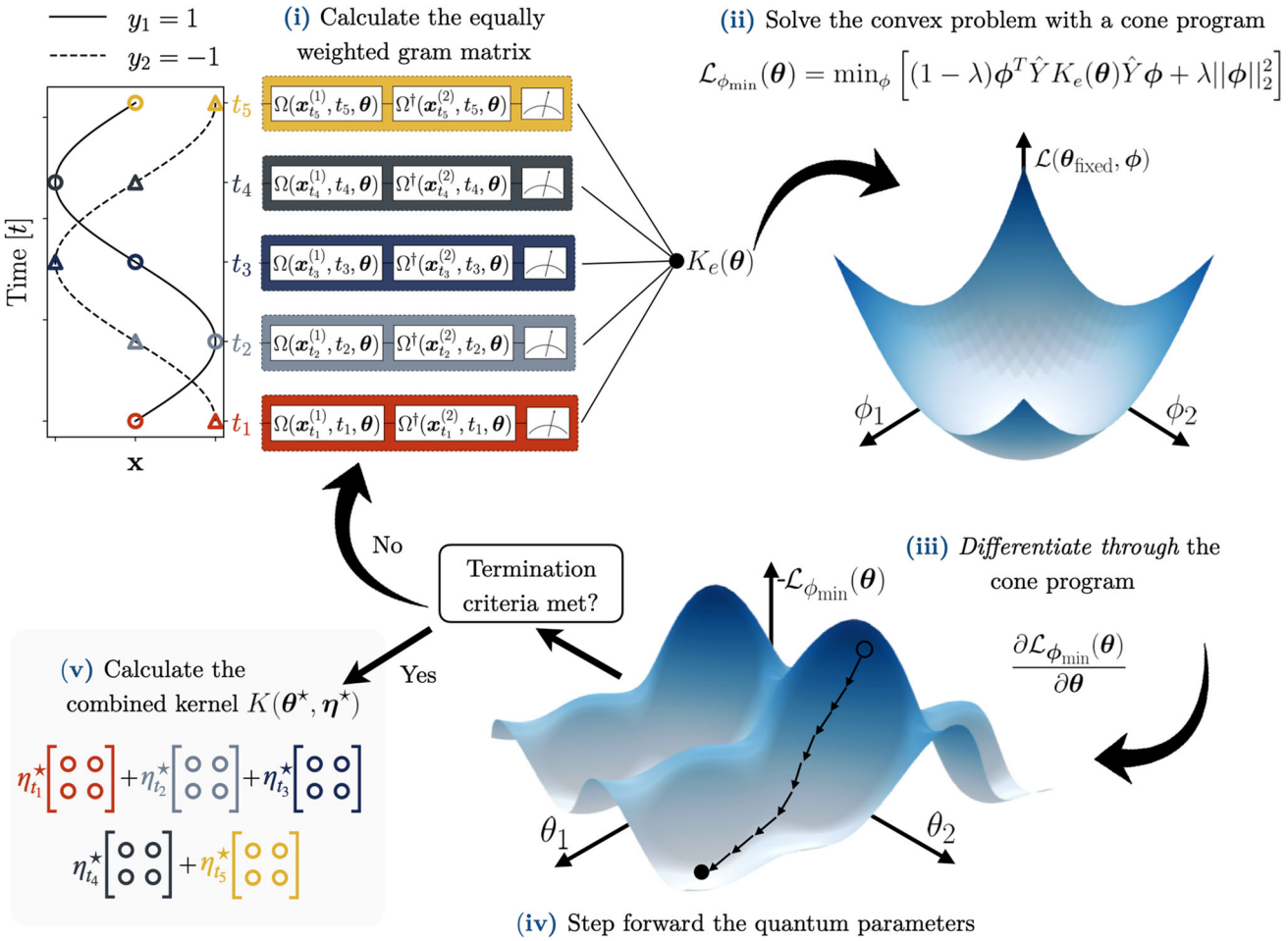

Fig. 1 Optimizing the combined quantum kernel for a simplified sinusoid versus cosinusoid classification problem with only *one training example* from class $y_i = 1$ $[-\sin(t)]$ and class $y_i = -1$ $[-\cos(t)]$. (i) Given some initial $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{init}}$, calculate the kernels of Eq. 5 at each time t_l and calculate the equally weighted Gram matrix $K_e(\boldsymbol{\theta}) = \sum_{t\in T} K_t(\boldsymbol{\theta})$. Unitary operator diagrams are color-coded in the figure and because there is only one unique element (excluding the diagonal which is unity by definition) only one unitary diagram is shown for each kernel computation. (ii) Solve the convex optimization problem with respect to the Lagrangian dual variables $\boldsymbol{\phi}$ using a cone program. (iii) Calculate the quantum gradients $\partial\mathcal{L}_{\phi_{\min}}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ by differentiating through the cone program. The surface plot shows a convex optimization landscape spanned by ϕ_1 and ϕ_2. (iv) Use the calculated $\partial\mathcal{L}_{\phi_{\min}}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ to step forward a derivative-requiring optimization routine (i.e., gradient descent, `Adam` etc.). If the termination criteria (a fixed number of loss iterations, convergence tolerance for $\mathcal{L}_{\phi_{\min}}$ etc.) are met with this new step, advance to (v) else, go back to (i) and enter new loop iteration. Note the negative sign of $\mathcal{L}_{\phi_{\min}}(\boldsymbol{\theta})$ in part (iv) of the figure. This is because in practice we maximize the function by minimizing its negative value. (v) Extract the optimal kernel weights $\boldsymbol{\eta}^\star$ using Eq. 11 to yield the combined kernel function (the TSHK) from Eq. 6. This kernel function can now be used to calculate kernel matrices $K(\boldsymbol{\theta}^\star, \boldsymbol{\eta}^\star)$ for use in any kernelized learning task including SVM

example, if we are given "windowed data" (sub-sequences taken from a larger time series), the initial condition is not guaranteed to be shared among data instances. One example of this is the classification of different genres of music given small audio clips from a longer song. It should be noted that even in this case, it could be possible to increase the learnability of the TSHK by preprocessing time series using a time-axis alignment protocol like Dynamic Time Warping (Sakoe Chiba 1978). The success of such an approach would have to be assessed on a case-by-case basis.

2.2 Training the kernel function

We now present a strategy for training the kernel function of Eq. 6 in a supervised learning setting. Throughout this section, Fig. 1 should be used as a companion and will be referenced throughout. To begin, we assume to have access to training set X containing $N_X \in \mathbb{Z}^+$ time-series instances $\boldsymbol{x}^{(i)}$ such that

$$X := \{\boldsymbol{x}^{(i)} : i \in \mathbb{Z}^+_{\leq N_X}\} \tag{7}$$

where each time-series instance $\boldsymbol{x}^{(i)}$ has a corresponding binary class label $y_i \in \{1, -1\}$. Using this labeled data, we define a suitable loss function to be optimized.

For optimizing the kernel coefficients $\boldsymbol{\eta}$, it was proposed for the EasyMKL algorithm (Aiolli Donini 2015) that the objective function from the Kernel Optimization of the Margin Distribution (KOMD) (Aiolli et al. 2008) approach should be used. The KOMD objective represents the separation between positive ($y_i = 1$) and negative ($y_i = -1$) training examples where the kernel weights are represented implicitly with the Lagrangian dual variables $\boldsymbol{\phi} := [\phi_1, \phi_2, \ldots \phi_{N_X}] \in \mathbb{R}^{N_X}$. That is, $\boldsymbol{\eta}$ is derivable using $\boldsymbol{\phi}$ as is shown later in Eq. 11 (see Aiolli Donini (2015) for explicit steps in transforming the optimization problem in $\boldsymbol{\eta}$ to that of $\boldsymbol{\phi}$ and vice versa). Recalling that our kernel function is also parameterized by the unitary operator parameters $\boldsymbol{\theta}$, the separation between classes as defined in Aiolli Donini (2015) is modified to be written as a function of both parameter sets $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) := (1 - \lambda)\boldsymbol{\phi}^T \hat{Y} K_e(\boldsymbol{\theta}) \hat{Y} \boldsymbol{\phi} + \lambda ||\boldsymbol{\phi}||_2^2 \tag{8}$$

where $K_e(\boldsymbol{\theta}) = \sum_{t \in T} K_t(\boldsymbol{\theta})$ is an equally weighted sum of Gram matrices with elements $K_t^{ij}(\boldsymbol{\theta}) = \kappa_t(\boldsymbol{x}_t^{(i)}, \boldsymbol{x}_t^{(j)}, \boldsymbol{\theta})$ from Eq. 5, $\hat{Y} = \text{diag}(y_1, y_2, \ldots, y_{N_x})$ and $\lambda \in [0, 1]$ is a real-valued penalty scaling factor hyperparameter intended to favor low variance solutions for $\boldsymbol{\phi}$ with increasing λ. As shown in Fig. 1(i), $K_e(\boldsymbol{\theta})$ is initially computed with some $\boldsymbol{\theta}_{\text{init}}$ (chosen randomly or otherwise) which is used to prime the first step of the max-min optimization problem

$$\mathcal{L}^\star := \max_{\boldsymbol{\theta}}[\min_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})] \tag{9}$$

where $\mathcal{L}^\star$ is the optimal loss and the optimal arguments are $\boldsymbol{\theta}^\star$ and $\boldsymbol{\phi}^\star$, respectively. Because the internal minimization with respect to $\boldsymbol{\phi}$ is a convex objective, as shown in Fig. 1(ii), for any given fixed $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{fixed}}$, we can efficiently and accurately find $\min_{\boldsymbol{\phi}}[\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})]$ using a convex solver such as a cone program (O'Donoghue et al. 2016a). Because we can reasonably assume that the global minimum of a convex problem will be found, we can re-frame the problem as

$$\mathcal{L}^\star = \max_{\boldsymbol{\theta}}[\mathcal{L}_{\boldsymbol{\phi}_{\min}}(\boldsymbol{\theta})] \tag{10}$$

where $\mathcal{L}_{\boldsymbol{\phi}_{\min}}(\boldsymbol{\theta}) := \min_{\boldsymbol{\phi}}[\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})]$ and the subscript $\boldsymbol{\phi}_{\min}$ indicates that we are working at the $\boldsymbol{\phi}$ which minimizes $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$ with respect to $\boldsymbol{\phi}$ for any given $\boldsymbol{\theta}$.

The maximization with respect to $\boldsymbol{\theta}$ can be performed using any general purpose optimization algorithm. Importantly, the gradient $\partial \mathcal{L}_{\boldsymbol{\phi}_{\min}}(\boldsymbol{\theta}) / \partial \boldsymbol{\theta}$ is obtainable analytically through an automatic differentiation engine (Bergholm et al. 2018) because (i) partial gradients of the kernel function with respect to the quantum circuit parameters $\boldsymbol{\theta}$ are known using techniques including the parameter shift rule and its variants (Mitarai et al. 2018; Schuld et al. 2019; Bergholm et al. 2018) and (ii) the partial gradients of the *minimum* of the convex optimization problem with respect to the quantum circuit parameters are known because the cone program is differentiable (Agrawal et al. 2019a, b) [Fig. 1(iii)]. This allows one to update the quantum parameters $\boldsymbol{\theta}$ [Fig. 1(iv)] using one of a whole host of gradient-requiring optimizers available from the wider ML literature (Bottou et al. 2018). Indeed, our method can be interpreted as a hybrid neural network with quantum layers and classical convex optimization layers (Agrawal et al. 2019a): a Quantum-Classical-Convex network (QCC-net).

After meeting the termination criteria of the optimization (which could be a fixed number of iterations or tolerance on changes in the value of Eq. 8 when the parameters are updated, for example), we arrive at an approximation of $\boldsymbol{\theta}^\star$ and $\boldsymbol{\phi}^\star$ where the latter is given as $\text{argmin}_{\boldsymbol{\phi}}[\mathcal{L}(\boldsymbol{\theta}^\star, \boldsymbol{\phi})]$. We use both optimal parameter vectors to extract the optimal kernel weights η_t^*

$$\eta_t^* = \frac{\boldsymbol{\phi}^{\star T} \hat{Y} K_t(\boldsymbol{\theta}^\star) \hat{Y} \boldsymbol{\phi}^\star}{\sum_{t \in T} \boldsymbol{\phi}^{\star T} \hat{Y} K_t(\boldsymbol{\theta}^\star) \hat{Y} \boldsymbol{\phi}^\star}. \tag{11}$$

Now, in Eq. 6, we set $\boldsymbol{\theta} \to \boldsymbol{\theta}^\star$ and $\boldsymbol{\eta} \to \boldsymbol{\eta}^\star$ to obtain our goal: a trained TSHK [Fig. 1(v)].

It should also be noted that during the training cycle, it is often appropriate to work with mini-batches of the training data. This both reduces the computational effort required in

training and is known to improve generalization performance (Keskar et al. 2017) in classical ML algorithms. Formally, a mini-batched iteration of the training cycle involves sampling, at random, a training mini-batch $X_{\text{batch}} \subset X$ with cardinality $|X_{\text{batch}}| = N_{\text{batch}}$. During this cycle, Gram matrices $K_t(\boldsymbol{\theta})$ and $K_e(\boldsymbol{\theta})$ and the diagonal label matrix $\hat{Y}$ have the reduced dimension $\mathbb{R}^{N_{\text{batch}}} \times \mathbb{R}^{N_{\text{batch}}}$ and the Lagrangian parameter vector $\boldsymbol{\phi}$ reduces its dimension to $\mathbb{R}^{N_{\text{batch}}}$. It should be clear now that the wall-time for a single mini-batch iteration using a fixed number of shots to evaluate the quantum kernel elements and a fixed number of iterations to solve the convex problem scales quadratically with $\mathbb{R}^{N_{\text{batch}}}$. When these batches are sampled, we must ensure that each mini-batch contains at least one time-series instance with label $y_i = 1$ and another with $y_i = -1$. Without this, we cannot define a separation between classes, and therefore the loss function of Eq. 8 is undefined.

Before moving on, we must consider the trainability of the TSHK. Among the key ingredients of the TSHK are of course quantum kernels. It is known that for static quantum kernels, quantum circuit gradients vanish exponentially with the number of qubits when classical data is embedded using Ansätze without sufficient inductive biases (Kübler et al. 2021). This is one of the manifestations of the barren plateaus phenomenon (McClean et al. 2018). Within our QCC-net framework, because quantum circuit gradients are obtained by differentiating through convex optimization layers, it is unclear to what extent QCC-nets suffer from barren plateaus. Indeed, not all quantum neural network architectures are found with this problem (Pesah et al. 2021). An interesting topic of future study would be a theoretical analysis of gradient scaling in QCC-nets and how to use convex optimization layers to impart different inductive biases.

2.3 Using the trained kernel function: classification

Now equipped with an optimal kernel function, we can use it in any kernelized ML technique like kernelized SVM or kernel ridge regression. In this work, we focus on classification using the kernelized soft margin variation of SVM which from now on-wards we refer to as "SVM." While a full presentation of the formalism describing SVM is beyond the scope of this work (and is discussed elsewhere (Cortes Vapnik 1995)), we now provide a high level explanation. In binary classification, SVM separates data points belonging to two different classes by learning a decision boundary, also known as a hyperplane, from labeled training data. It allows for some misclassifications by introducing a soft margin, where the appetite for misclassification is controlled by a single hyperparameter $C \in \mathbb{R}$. A kernel function is used to map the data into a higher-dimensional space where the hyperplane can separate the classes more easily.

For our purposes, SVM can be regarded as a decision function for an unseen data point $\boldsymbol{x}$ given four items: $\kappa(\boldsymbol{x}, \boldsymbol{x}')$, X, $\boldsymbol{y}$, and C where each, in order, is a valid kernel function, a set of training data, the vector of binary class labels for that training data, and the misclassification appetite hyperparameter. That is, we have

$$\mathcal{D}(\boldsymbol{x}|\kappa(\boldsymbol{x}, \boldsymbol{x}'), X, \boldsymbol{y}, C) := \mathcal{D}^{\star}(\boldsymbol{x}) \in \mathbb{R}. \tag{12}$$

Binary class predictions can now be made using

$$y_{\text{pred}}(\boldsymbol{x}) = \text{sgn}[\mathcal{D}^{\star}(\boldsymbol{x})] \in \{-1, 1\}. \tag{13}$$

The most natural way to use this framework to perform time-series classifications is simply to set $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ to a trained kernel function of the form of Eq. 6 at the optimal parameters $\boldsymbol{\theta}^{\star}$ and $\boldsymbol{\eta}^{\star}$. Another way of doing so is to break the problem down into p-many time-dependent SVMs. We then have

$$\mathcal{D}_t(\boldsymbol{x}_t|\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}_t', \boldsymbol{\theta}^{\star}), X_t, \boldsymbol{y}, C) := \mathcal{D}_t^{\star}(\boldsymbol{x}_t) \in \mathbb{R} \tag{14}$$

where $X_t = \{\boldsymbol{x}_t^1, \boldsymbol{x}_t^2, \ldots \boldsymbol{x}_t^{N_X}\}$. This function is then used to make a prediction at each t with $y_{\text{pred}}^t(\boldsymbol{x}_t) = \text{sgn}[\mathcal{D}_t^{\star}(\boldsymbol{x}_t)]$. The final class prediction for the entire time-series $\boldsymbol{x}$ is then given by a weighted majority vote from each t

$$y_{\text{pred}}^{\text{vote}}(\boldsymbol{x}) = \text{sgn}\left[\sum_{t \in T} \eta_t \cdot y_{\text{pred}}^t(\boldsymbol{x}_t)\right] \in \{-1, 1\}. \tag{15}$$

While it is not guaranteed that $y_{\text{pred}}(\boldsymbol{x})$ will provide more accurate predictions than $y_{\text{pred}}^{\text{vote}}(\boldsymbol{x})$ in general, we note that because our training procedure defined in Section 2.2 refines a combined kernel [used to define $y_{\text{pred}}(\boldsymbol{x})$], it is intuitive to expect this. Indeed, we find that $y_{\text{pred}}(\boldsymbol{x})$ is most effective for the data sets treated in Sects. 4.1, 4.2, and 5.3, so we choose to make predictions using Eq. 13. However, analyzing $\mathcal{D}_t^{\star}(\boldsymbol{x}_t)$ and $y_{\text{pred}}^t(\boldsymbol{x}_t)$ can provide valuable insights into the time-dependent nature of a given TSHK function which *are used* in this work.

3 Probing time dependence

After having achieved a reasonable approximation of $\boldsymbol{\theta}^{\star}$ using a classical optimization routine, a relevant quantity to probe is the time dependence of the inner product space. This is an important question to ask because while usage of the TSHK allows for a time dependent inner product space, it does not mean that strong time dependence was achieved when minimizing the cost function of Eq. 8 which will be strictly dependent on the data set and the choice of parameters/circuit Ansätze used to construct the QCC-net. As an extreme case, there can exist some $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ such that

$e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t} \approx \mathbb{I} \quad \forall t$ thus the resulting TSHK is approximately time-independent. Naively, one may initially think to examine the value of $\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta}^\star)$ as a function of t where $\boldsymbol{x}_t$ and $\boldsymbol{x}'_t$ are drawn from a testing data set (or iterate over an entire testing set to calculate an entire kernel matrix). While there is nothing wrong with this approach, we must note that in the ideal case from training, time dependence in this sense *will disappear*. That is, at each point in time $t \in \mathcal{T}$, the kernel function was trained to embed classical data with label $y_i = 1$ in a separate region of Hilbert space to those with label $y_i = -1$. In the ideal case, these two regions do not overlap at all, and hence $\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta}^\star) = 0 \; \forall t$ when $\boldsymbol{x}_t$ and $\boldsymbol{x}'_t$ belong to different classes. Before the kernel function is trained, finite values of $\kappa_t(\boldsymbol{x}_t, \boldsymbol{x}'_t, \boldsymbol{\theta})$ are of course permitted under the same conditions for the input. The above discussion is best explained alongside Fig. 2. Intended for illustrative purposes only, Fig. 2 compresses high-dimensional vectors $|v\rangle \in \mathbb{C}^{2^n}$ into two dimensional regions on the surface of a sphere. Differently colored regions represent the space of feature vectors $|\boldsymbol{x}_t, t, \boldsymbol{\theta}\rangle$ accessible in different domains defined by the classical data $\boldsymbol{x}_t$ (see the figure caption for more information). Looking at Fig. 2a, we can see that at any t, the time-dependent Hilbert space regions $\mathcal{H}_t^{y_i=1}$ do not overlap with $\mathcal{H}_t^{y_i=-1}$ when the kernel function is ideally trained (regions marked "T"). For an untrained kernel (or at least not ideally trained kernel; regions marked "U"), overlap between these regions are permitted.

To provide a more intuitive notion of time dependence, we must ask another question: "given a general classical data vector $\boldsymbol{\chi} = [{}^1\chi, {}^2\chi, \ldots {}^d\chi] \in \mathbb{R}^d$, does the quantum state embedding of $\boldsymbol{\chi}$ given by Eq. 2 when we set $\boldsymbol{x}_t \to \boldsymbol{\chi}$ change as $t \to t + \delta t$?".

The quantity of interest to probe this time dependence is the overlap of the t and $t + \delta t$ quantum embedding spaces integrated over all $\boldsymbol{\chi}$. That is

$$f_{\boldsymbol{\theta}}(\delta t) := \int_{d\chi_{\text{lo}}}^{d\chi_{\text{hi}}} \cdots \int_{1\chi_{\text{lo}}}^{1\chi_{\text{hi}}} |\langle \boldsymbol{\chi}, t, \boldsymbol{\theta} | \boldsymbol{\chi}, t+\delta t, \boldsymbol{\theta}\rangle|^2 d^1\chi \ldots d^d\chi \tag{16}$$

where the limits ${}^m\chi_{lo}$ and ${}^m\chi_{hi}$ are the minimum and maximum values permitted for classical data embedded into the chosen quantum feature space, respectively. Many quantum feature maps are periodic such that ${}^m\chi \in [{}^m\chi_{\text{lo}}, {}^m\chi_{\text{hi}})$ where ${}^m\chi_{\text{lo}} = 0$ and ${}^m\chi_{\text{hi}} = 2\pi$. Should either limit be unbound, limits should be set to enclose the zone of interest (i.e., some region enclosing all training and testing examples) to the data set. After simplifying the integrand using Eq. 2 (namely, $U^\dagger(\boldsymbol{\chi}, \boldsymbol{\alpha})U(\boldsymbol{\chi}, \boldsymbol{\alpha}) = \mathbb{I}$ and canceling t in the exponent of the

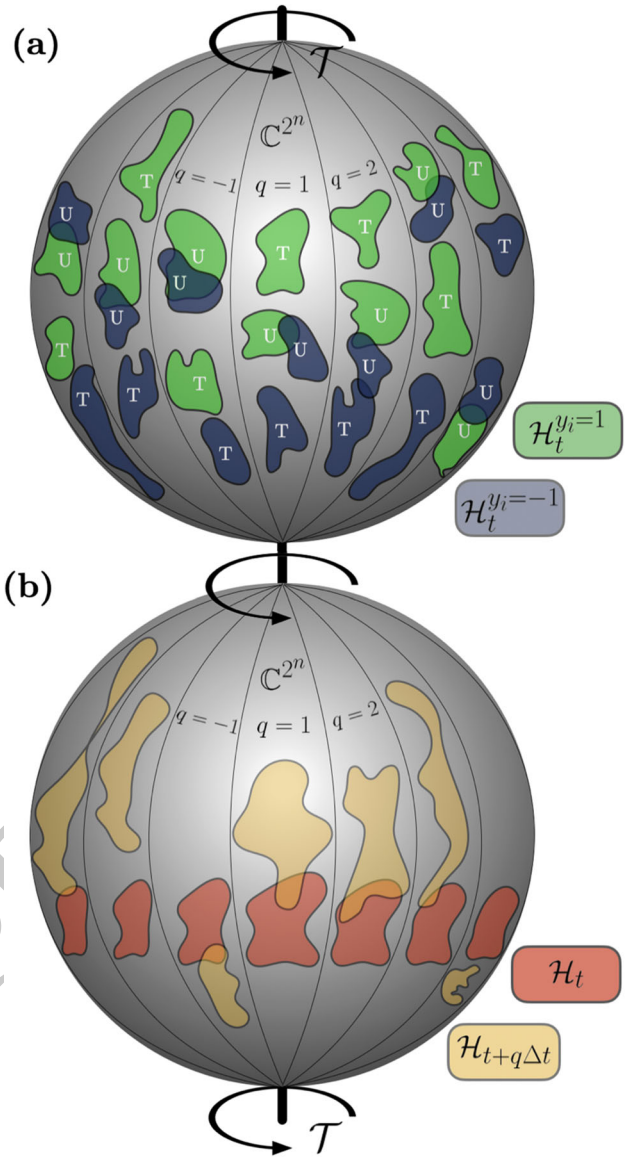

Fig. 2 Time-evolving Hilbert space illustrations. Spheres are divided into Q-many petals each indexed by $q \in \mathbb{Z}^{\neq 0}$. In each petal, the grey area represents the vector space spanning $\mathbb{C}^{2^n}$ such that each point on a petal represents a vector $|v\rangle \in \mathbb{C}^{2^n}$. A full rotation of the sphere corresponds to the period $\mathcal{T}$ of time evolution operator $e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})t}$. We note that a period $\mathcal{T}$ formally exists when the spectrum of H is free of irrational eigenvalues. The $q = 0$ case is omitted to retain visual clarity in **b** where colored regions would now (by definition of Eqs. 16 and 17) maximally overlap. **a** Hilbert space regions accessible by embedded classical data belonging to class 1 or -1 where the kernel function has been Trained (T) or is untrained (or at least not ideally trained; U) at different times $t = t_q$. **b** Hilbert space regions accessible by the entire classical data input space $\boldsymbol{\chi} \in \mathbb{R}^d$ at time t and time $t + q\Delta t$ where Δt is a fixed time step

time evolution operator), we find

$$f_{\boldsymbol{\beta},\boldsymbol{\gamma}}(\delta t) := \langle a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma} | P_0 | a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle \prod_{j=1}^{d} ({}^{j}\chi_{hi} - {}^{j}\chi_{lo}) \quad (17)$$

where $|a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle := e^{-iH(\boldsymbol{\beta},\boldsymbol{\gamma})\delta t}|0\rangle^{\otimes n}$ and $P_0 = |0\rangle^{\otimes n}\langle 0|^{\otimes n}$. Noting also that the result of the product over the index j in Eq. 17 is a real constant scaling factor k, it is useful to set $k = 1$ choosing to work with the quantity $\mathcal{F}_{\boldsymbol{\beta},\boldsymbol{\gamma}}(\delta t) = \langle a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma} | P_0 | a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle \in [0, 1]$. Examining $\mathcal{F}_{\boldsymbol{\beta},\boldsymbol{\gamma}}(\delta t)$, it is clear that we need only to consider the time evolution operator acting on the zero state with no consideration of classical data points. Indeed, much like Eq. 5, $\mathcal{F}_{\boldsymbol{\beta},\boldsymbol{\gamma}}(\delta t)$ can be efficiently computed by estimating the probability of measuring the bit-string of all zeros in the computational basis when a quantum computer is prepared in the state $|a_{\delta t}, \boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$.

The meaning of Eq. 16 (or, equivalently, Eq. 17) is demonstrated visually in Fig. 2b. The sphere is constructed in the same manner as Fig. 2a, but colored regions now represent the Hilbert space region accessible by the embedding of $\boldsymbol{\chi}$ at time t ($\mathcal{H}_t$; red regions) and at time $t + q\Delta t$ ($\mathcal{H}_{t+\Delta t}$; yellow regions) where $q \in \mathbb{Z}^+$ and $\Delta t \in \mathbb{R}_{>0}$ are fixed constants. The squared modulus of overlapping regions of red and yellow in Fig. 2b can be interpreted as either Eq. 16 or Eq. 17.

4 Demonstrations with quantum circuit simulators

4.1 Building intuition: a synthetic example

As a first example of our time-series classification algorithm, we deal with a didactic case to build intuition. Our goal is to create a synthetic time-series classification problem where (i) data is not linearly separable (thus kernelized SVM is required) and (ii) the shape and position of the decision boundary must evolve with time in order to successfully classify data at different values of t. Accordingly, we generate a training set X_{syn} with $|X_{\text{syn}}| = 100$ two-dimensional ($d = 2$) time-series instances. We construct the beginning and endpoints of each time series using the well-known moons and circles data sets, respectively. Intermediate points in the series are generated using a 10-step (i.e., $p = 10$) linear interpolation between the start and end points. The result is a gradual transition from moons to circles which we call the `moons2circles` data set. Testing data from `moons2circles` is shown as markers overlaid on the panels of Fig. 3a.

We use two different varieties of $n = 3$ qubit quantum circuit structures to implement the quantum components of the QCC-net. We denote first variety R_y-SEL-g and the second QAOA-g-SEL-g. We implement $g = 1$ and 3 varieties where g denotes the number of repeated layers of the layered Ansätze. R_y means a fixed embedding circuit (i.e., $\boldsymbol{\alpha}$ is dropped) of $U(\boldsymbol{x}_t) = R_y({}^1x_t) \otimes R_y({}^2x_t) \otimes I$ is used and QAOA means an embedding circuit inspired by the Quantum Alternating Operator Ansätz (Lloyd et al. 2020; Farhi et al. 2014; Hadfield et al. 2019) is used to implement $U(\boldsymbol{x}_t, \boldsymbol{\alpha})$. SEL means a Strongly Entangling Layers Ansätz (Schuld et al. 2020) is used to implement $W(\boldsymbol{\beta})$. For all circuits, $D(\boldsymbol{\gamma}, t)$ is implemented using a circuit representing a truncated n-local Walsh-operator expansion as described in Welch et al. (2014). The kernel function of Eq. 6 is then trained using the process described in Section 2.2 for 250 mini-batch iterations with $N_{\text{batch}} = 4$ using the `Adam` optimizer (Kingma Ba 2017) with an initial learning rate of 0.05. For solving the convex problem (i.e., finding the Lagrangian dual variables $\boldsymbol{\phi}$), we use the splitting conic solver (O'Donoghue et al. 2016b; O'Donoghue 2021) which interfaces `cvxpyayers` (Agrawal et al. 2019a) for gradient computations (Agrawal et al. 2019b) which are passed to the automatic differentiation in Pennylane (Bergholm et al. 2018). The training is repeated 50 times where each run is seeded with initial parameters $\boldsymbol{\theta}_{\text{init}}$ drawn uniformly at random from the range $[-\pi, \pi]$.

The results of training are shown in Fig. 3b–e. On average, we can see that QAOA-3-SEL-3 produces the largest loss values both initially (i.e., at the first mini-batch iteration) and after training for the 250 mini-batch iterations. While it is intuitive that the loss after training should be largest for QAOA-3-SEL-3 simply because it has the largest number of quantum circuit parameters $\boldsymbol{\theta}$, this does not explain the large initial loss. Indeed, at the first mini-batch iteration, we measure the separation of binary classes for optimally combined random kernels. This means that combined random kernels become better separators as the depth and number of parameters increases for the `moons2circles` data set. This effect is so pronounced that *untrained* QAOA-3-SEL-3 models, on average, have larger losses than trained R_y-SEL-1 and R_y-SEL-3 models. Examining the density heat maps (grey shaded areas on Fig. 3b–e), we can see that training is highly stochastic, much of which is owed to the small mini-batch size. We do see this stochasticity reduce for the QAOA-g-SEL-g models where it is more obvious that an envelope of high density forms around the mean curve showed in dark red.

Using the best (as determined by the largest value of the loss evaluated using the entire testing data set) trained TSHK with the QAOA-3-SEL-3 Ansätz, we train a SVM with $C = 100$ at each of the 10 time steps and compute $y^t_{\text{vote}}(\boldsymbol{x}_t)$ on a fine 100×100 grid at each t. These results are shown on the upper panel of Fig. 3a as heat-maps at each point in time. Regions colored in dark red/dark blue show areas where $y^t_{\text{vote}} = 1/\ y^t_{\text{vote}} = -1$ The result is clear: the decision function is time-evolving and adapts to separate data belong-

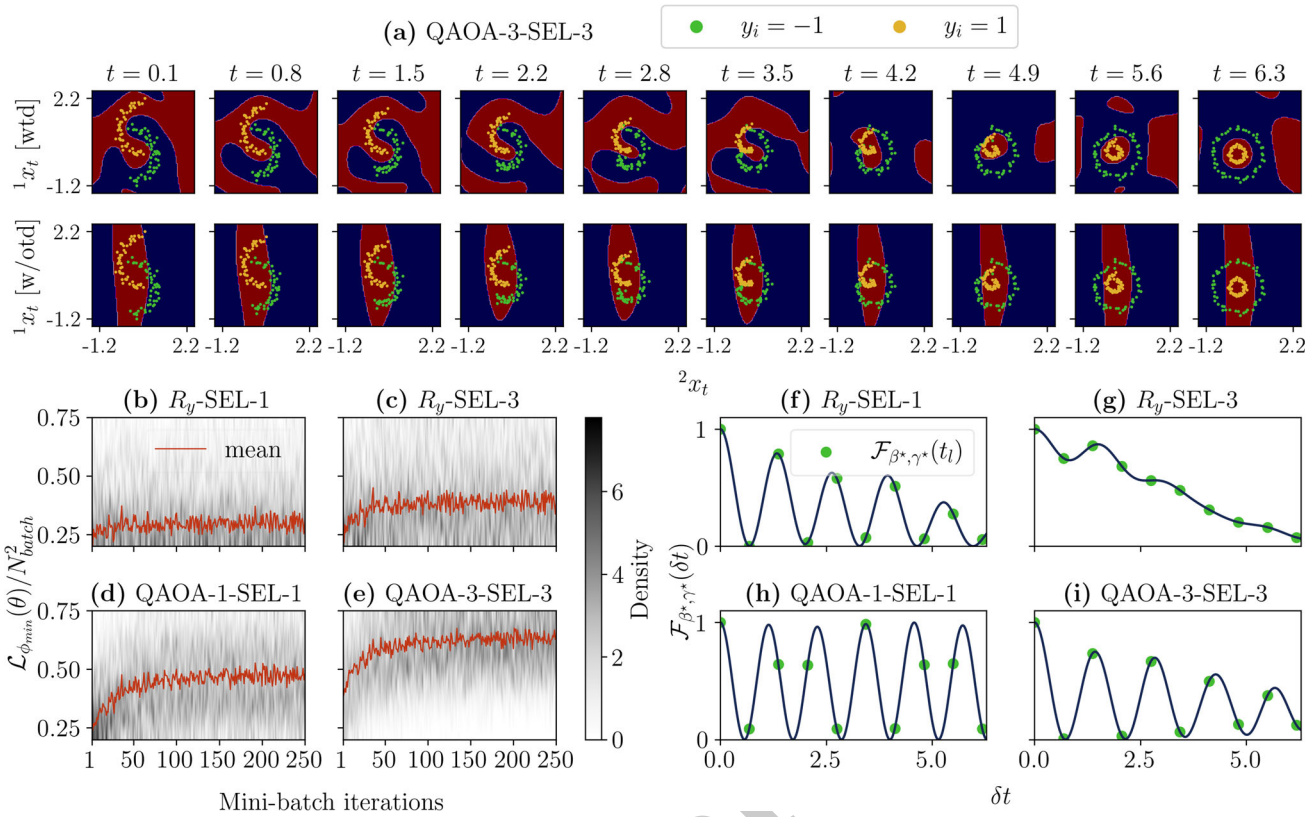

Fig. 3 Quantum circuit simulator experiments with the `moons2circles` data set. **a** The evolution of the time-dependent class prediction y^t_{vote} as moons (far left panel) gradually become circles (far right panel). Regions predicting $y_i = 1$ are shown in dark red and regions predicting $y_i = -1$ are shown in dark blue. The Testing data set is overlaid and are colored by their truth values as given in the legend. The figure is generated using the best model obtained for the QAOA-3-SEL-3 runs described in the main text. The upper panel displays results with time dependence (wtd) in the inner product space, while the lower panel displays results without time dependence in the inner product space (w/otd) (**b**–**e**). The normalized loss $\mathcal{L}_{\boldsymbol{\phi}_{\text{min}}}(\boldsymbol{\theta})/N^2_{\text{batch}}$ as a function of mini-batch iterations for the R_y-SEL-1, R_y-SEL-3, QAOA-1-SEL-1, and QAOA-3-SEL-3 quantum circuits, respectively. The mean value of 50 independent runs is shown in red and the probability density over these runs is shaded in grey. Recall that according to Eq. 10 we are solving a *maximization* problem, so we expect the loss in *increase* with the number of mini-batch iterations. **f**–**i** For the same quantum circuits and order defined in **b**–**e**, the time-resolved embedding overlap $\mathcal{F}_{\boldsymbol{\beta^\star},\boldsymbol{\gamma^\star}}(\delta t)$ is plotted using the best model obtained using each circuit. Green markers are inserted at $\mathcal{F}_{\boldsymbol{\beta^\star},\boldsymbol{\gamma^\star}}(t_l)$ to show overlaps of embedding spaces where classical data was seen in training

ing to the two different classes at every time step. Although this is a simple example, we note that the best TSHK for any circuit structure variation achieves 100% classification accuracy when passed to a SVM. This is despite the data at intermediate time steps not being perfectly separated (data of different classes overlap). Upon inspection of the kernel weights, we see that the beginning and end-points are most heavily weighted which is intuitive since these points see no overlap between classes. This observation exposes the power of our algorithm: time-series can be classified based on "smoking-gun" time points as these points are able to be heavily weighted by the convex optimizer.

The effects of including a time-dependent inner product space become even clearer should we observe the performance on the `moons2circles` data set in its absence. This is shown in the lower panel of Fig. 3a as achieved by fixing $t = 1$ (an arbitrary choice) in the time evolution operator. Other settings are identical to those which produce the results of the upper panel of Fig. 3a. While there remains some change in the position of the decision boundary as we advance in time, it is certainly much less malleable than in the case with the time dependent inner product. Now unable to use a different kernel function at each t, training finds a single kernel function which strikes a balance between being able to classify moons-like and circles-like data, which, when inserted into a SVM, results in decision boundaries with a striped character. Moreover, classification accuracy with this model slips down to 78%. From these investigations, we conclude that using the TSHK is most effective for data sets where (i) the correct notion of similarity between pairs of data points changes with time and (ii) there exists a shared time axis in the data as was described in Section 2.1.

In Fig. 3f and g, we probe the time dependence of the best trained models using the approach presented in Section 3. With the exception of R_y-SEL-3 (Fig. 3g), $\mathcal{F}_{\boldsymbol{\beta}^\star,\boldsymbol{\gamma}^\star}(\delta t)$ is observed to be highly oscillatory. Importantly, apart from QAOA-1-SEL-1 (Fig. 3h), $\mathcal{F}_{\boldsymbol{\beta}^\star,\boldsymbol{\gamma}^\star}(\delta t)$ is a broadly decreasing function intuitively suggesting that the quantum embedding space when data is more moons-like becomes different as the data becomes more circles-like. Indeed, even for QAOA-SEL-1, the last point (shown with green markers) of all models shows that $\mathcal{F}_{\boldsymbol{\beta}^\star,\boldsymbol{\gamma}^\star}(\delta t)$ commensurate with the time difference between exactly moons (the left-most panel of Fig. 3a) and exactly circles (the right-most panel of Fig. 3a) is very small which means that the quantum embedding space used for classifying moons is *strongly* different than that used for classifying circles.

4.2 Univariate time-series: the gun-point data set

In this section, we demonstrate the performance of our algorithm for classification on a real and popular benchmark: the gun-point data set (Ratanamahatana Keogh 2005). Although this data set has been described in detail elsewhere, we summarize briefly here. Two participants, with their hands beginning at their sides, are asked to either (i) point their right finger towards a target or (ii) point a small firearm (a hand gun) at a target and bring their hand back down to the starting position. During this process, the forward motion on their hand is tracked with a sensor at regular intervals and each finger-point or gun-point instance is recorded as a univariate ($d = 1$) time-series instance. Labeled gun-point instances belong to class 1 ($y_i = 1$), and finger-point instances belong to class 2 ($y_i = -1$). The training set X_{tr} has size $|X_{tr}| = 50$, and the testing set X_{te} has size $|X_{te}| = 150$. Each instance has 150 time stamps ($p = 150$).

Using the QAOA-3-SEL-3 circuit structure described in Section 4.1, we train four models with $n = 2, 4, 6$, and 8 qubits. Each model is selected from 20 training attempts each using 500 mini-batch iterations with $N_{\text{batch}} = 4$ with $\boldsymbol{\theta}_{\text{init}}$ drawn uniformly at random from $[-\pi, \pi]$. Each model is selected based on the largest value of the loss function evaluated on the testing data set. Before the SVMs are trained, given $\boldsymbol{\theta}^\star$ each optimal model is passed through the convex optimization step a final time using the entire training data set (i.e., $N_{\text{batch}} = N_X$) to refine the kernel weights $\boldsymbol{\eta}$. We now train $C = 100$ SVM decision functions $\mathcal{D}^\star(\boldsymbol{x})$ which we use to classify each time-series instance in X_{test}. With these predictions, we calculate the F_1 score, the balanced accuracy score (A_B), and the receiver operating characteristic area under the curve score (ROC AUC; abbreviated further to RA for the rest of this work).

Figure 4 shows the results of these experiments. In Fig. 4a, the time-series instances which were identified as the most class 1 and the most class 2 as determined by having the highest and lowest values of $\mathcal{D}^\star(\boldsymbol{x})$ are shown. Directly below on the same time axes, in Fig. 4b, we have the time-resolved kernel weights η_t for the $n = 2$ and $n = 8$ models. It is clear that different points in time are weighted very differently to others. In particular, peaks near the regions where the finger/gun is being lifted and peaks near where the finger/gun is being put back down again are seen. This is in stark contrast to the beginning, end, and middle points of the series where finger/gun is approximately stationary. This points to the key discriminating factors between the two cases being a reaction time difference and/or other characteristic differences during the time stamps where the finger/gun is *in motion*. It should also be noted that there is a remarkable symmetry in the profile of η_t about the mid-point of time (where the finger/gun is stationary); finger/gun instances can be discriminated equally well from motion on the way up to pointing at the target as they can on the way down. All of the models produce a similar profile for η_t, which, for $n = 2$ and $n = 8$, are shown in Fig. 4a. It can be seen that increases in the qubit size for the model lead only to small changes in the weighting of different time stamps in the classification. Looking now at Fig. 4c, we see that the balanced accuracy score is already at 97.2% for the $n = 2$ model, and raises a small amount to 98.0% by the time we reach $n = 8$. Similar small gains are seen for F_1 scores and the ROC AUC score. Remarkably, these scores mean that our hybrid quantum-classical algorithm is competitive with purely classical algorithms for this data set. Specifically, when compared against 9 state-of-the-art deep learning approaches (Fawaz et al. 2019), our algorithm is beaten in terms of accuracy scores by only two of them: fully convolutional neural networks (FCN; $A_B = 1.000$) and residual neural networks (ResNet; $A_B = 0.991$) which, in comparison, require far greater computational resources.

To quantify which components of our approach are most significant in our experiments and what training hyperparameters yield the best results, we conduct a further study focusing on the $n = 2$ qubit case. A summary of the results is shown in Fig. 5, and an expanded version of Fig. 5 is available in the Supplementary Information. In general, we find that training with small batch sizes and small λ is the most effective (Fig. 5a). This effect is enhanced when paired with the full TSHK formalism (red bars) which, for these hyperparameters, yields the largest A_B and F_1 scores as well as the largest training kernel alignment $\mathcal{A}_{train}$ (see caption of Fig. 5 for a definition) of all approaches taken in this study. This peak performance must, however, be viewed in the context of the other bars in Fig. 5a. Examining the case where the time dependent inner product is forbidden (grey bars), the full approach offers only a 1% improvement in A_B and F_1. Indeed, examining the general level of scores for all other bars on both Fig. 5a and b, we can see that any of the taken approaches can score in the mid-90% range. It must therefore be concluded that most of the performance we achieve on this

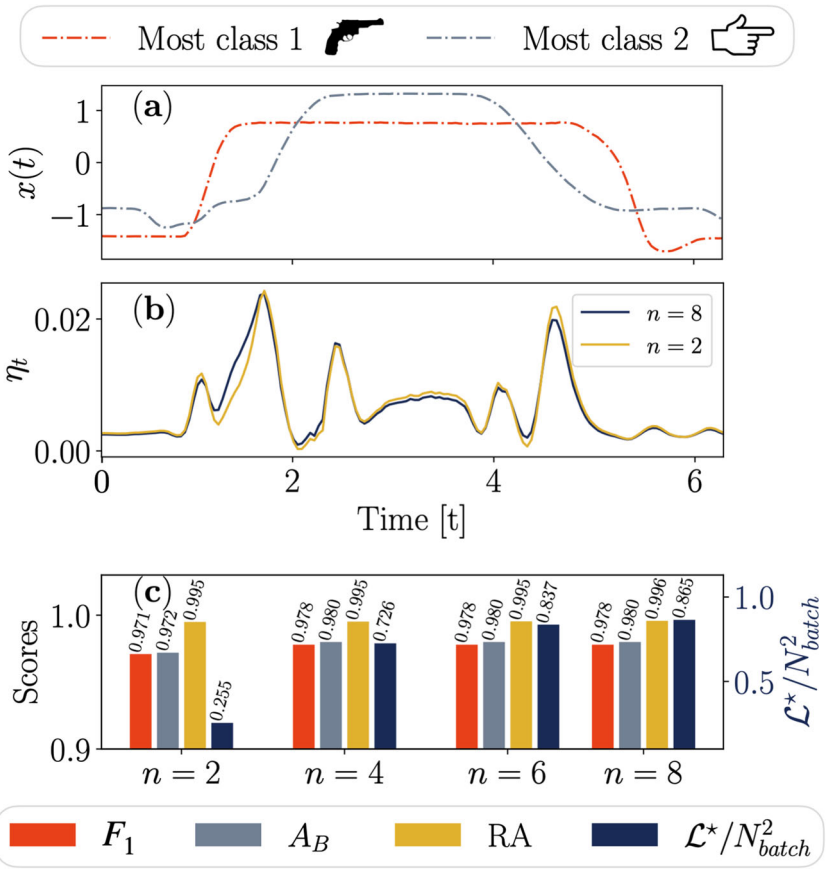

Fig. 4 Insights and performance metrics for trained TSHKs passed to SVMs on the gun-point data set. **a** Testing data set time-series instances reported to be the most class 1 (gun-point; orange dotted and dashed line) and most class 2 (finger point; blue dotted and dashed line) as determined by the value of $\mathcal{D}^\star(\boldsymbol{x})$. **b** The kernels weights η_t at each time for the $n = 2$ (gold line) and $n = 8$ (blue line). **c** F_1, A_B, and RA scores (left axis) and the normalized optimal loss $\mathcal{L}^\star/N^2_{batch}$ (right axis) for the $n = 2, 4, 6$, and 8 models

data set is gained from the classical portions of the algorithm (mostly the SVM itself) correctly classifying many instances using any of the combined kernels we supply it. Only a few extra points are gained by using the full TSHK formalism. Given how closely performant the top classical deep learning approaches are for this data set (Fawaz et al. 2019), it is, however, not unreasonable to suggest that these gains are indeed significant. Furthermore, we view the results presented in Fig. 4 to be sensitive to the choice of circuit Ansätz. It is unclear what choice should be made when presented with classical data and is a topic of current research to figure out how to encode inductive biases into classical data encoding Ansätze (Bowles et al. 2023).

Lastly, we remark that while scores only increase modestly as the number of qubits increases (Fig. 4c), the loss does gradually get larger. Recalling that this loss represents the separation between positive and negative classes, this means that, within the parameters of our study, higher qubit models are able to find a time-dependent quantum embedding space where classical data is better separated than smaller qubit models. This larger separation is known to give rise to higher generalization performance for SVM (Hastie et al. 2009).

5 Parallel execution on quantum hardware

5.1 Quantum multi-programming

To more efficiently use NISQ hardware, it is possible to run multiple quantum circuits concurrently. Known as Quantum Multi-Programming (QMP), this technique allows NISQ devices to execute several quantum circuits concurrently, even if they differ in structure or complexity. In this section, we show that computation of TSHKs can be significantly sped up using QMP.

The primary motivation for QMP is driven by the fact that the number of qubits present in NISQ devices is often significantly higher than their Quantum Volume (QV). Specifically, when compared to trapped ion devices, superconducting quantum computers have restricted qubit connectivity and

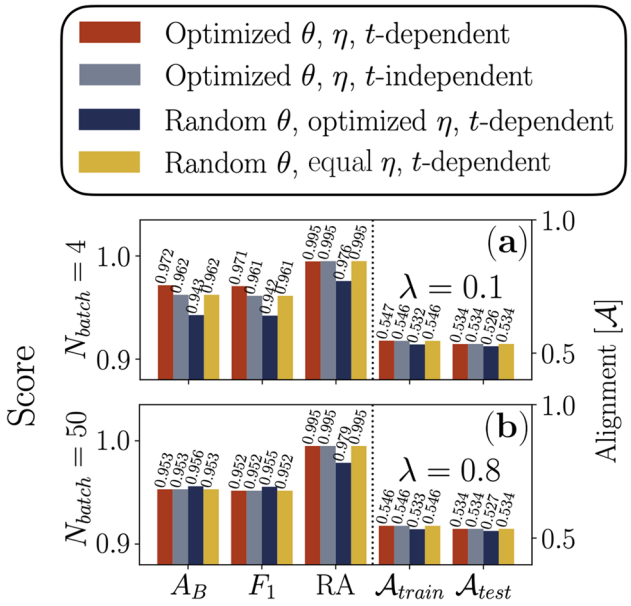

Fig. 5 Discerning the impact of different mechanisms defining the TSHK as applied to SVM classification of the gun-point data set. Different approaches include the full TSHK method (red bars), the otherwise full method with time dependence in the inner product space forbidden (a single static quantum kernel is used at each t; grey bars), the otherwise full method but with randomly chosen quantum circuit parameters $\boldsymbol{\theta}$ (dark blue bars) and an approach using random $\boldsymbol{\theta}$, equal kernel weights ($\eta_t = 1/p \forall t$) and a time-dependent inner product space. $\mathcal{A}_{train/test} = 1 - 1/p^2 \sum_i^p \sum_j^p |K_{train/test,ij} - K_{truth,ij}|$ is the kernel alignment of the training/testing data sets, $K_{train/test,ij}$ are the elements of combined kernels for the training/testing datasets and $K_{truth,ij} = y_i y_j$ are the elements of the truth matrix. **a** $N_{batch} = 4$ and $\lambda = 0.1$. **b** $N_{batch} = 50$ (the entire training data set) and $\lambda = 0.8$. An expanded version of this figure is available in the Supplementary Information

lower QV. Taking this further, following IBM's quantum volume V_Q, we have $\log_2 V_Q = \text{argmax}_m \{\min[m, d(m)]\}$ where $d(m)$ is the depth of a model circuit with m-qubits (Cross et al. 2019). One depth of the model circuit is composed of a random permutation of the qubits involved in the test, followed by random two-qubit gates. For instance, the QV of the `ibm_washington` device (used in this work) is 64, with 127 qubits available. This means the model circuit with 6-qubits runs reliably at six depths of the model circuit on average on `ibm_washington`. Other superconducting quantum computers share similar properties such as limited connectivity between qubits and many more qubits than $\log_2 V_Q$. Therefore, it is crucial to use the capacity of modern superconducting quantum devices more efficiently, and QMP has been proposed as a method to achieve this goal. Therefore, QMP fills the gap between the relatively many qubits and the relatively low QV of NISQ devices by executing multiple quantum circuits concurrently, enhancing the throughput and utilization of NISQ devices.

However, implementing QMP on NISQ devices has several issues to address because QMP accompanies unfavorable impacts on the whole system such as measurement timing of the concurrent circuits (Das et al. 2019) and crosstalk between different circuits (Ohkura et al. 2022). Efficiently mapping qubits between logical and physical states, as well as task scheduling (Liu Dou 2021; Niu Todri-Sanial 2021, 2022), have been studied alongside the aforementioned issues. Despite preliminary investigations into QMP, its integration with quantum algorithms, and especially QML, remains underdeveloped. Nonetheless, there has been recent noteworthy progress in applying QMP to Grover's search algorithm (Park et al. 2023), resulting in improved success probabilities compared to previous attempts.

5.2 QMP implementation considerations

Before proceeding to calculate TSHKs with QMP, we discuss the typical issues arising out of QMP. Starting with crosstalk, this effect is studied in detail by P. Murali et al. (2020). They suggested several rules to mitigate crosstalk between qubits. The relation between the number of physical buffers and the error rate was studied by Ohkura et al. (2022). They introduced a physical buffer, which is the number of idle qubits between quantum circuits. They tested a different number of controlled-X (CX) gates with a different number of physical buffers. Ohkura et al. (2022) concluded that only 1 physical buffer is sufficient until 30 CX gates. Therefore, we use one physical buffer for our implementations.

One important issue of QMP is measurement timing. When quantum circuits having different circuit depths run concurrently, the measurement timing influences the shortest circuit. This effect was studied in Das et al. (2019); Ohkura (2021). In these studies, they suggested delaying shorter circuits to align the measurement timing. In our case, because the circuits implementing elements of the TSHK each have the same depth, we do not need to make any alterations to measurement timings.

Another crucial issue is efficient physical qubit mapping. Systematic qubit mapping algorithms have been developed (Das et al. 2019; Liu Dou 2021; Niu Todri-Sanial 2022, 2021). Even though these qubit mapping algorithms are also important for quantum circuits executed in serial (i.e., without QMP), efficient qubit mapping becomes more challenging and crucial in QMP because practical QMP circuits use more qubits than serial circuits. In our particular circumstance where each quantum thread is composed of two logical qubits, the mapping problem is solved by hand by mapping each thread to two physical qubits with two-qubit gate connectivity subject to the constraint that *at least* one buffer qubit separates individual threads. The QMP layout can be seen in Fig. 6 for `ibm_washington` and `ibm_sherbrooke`.

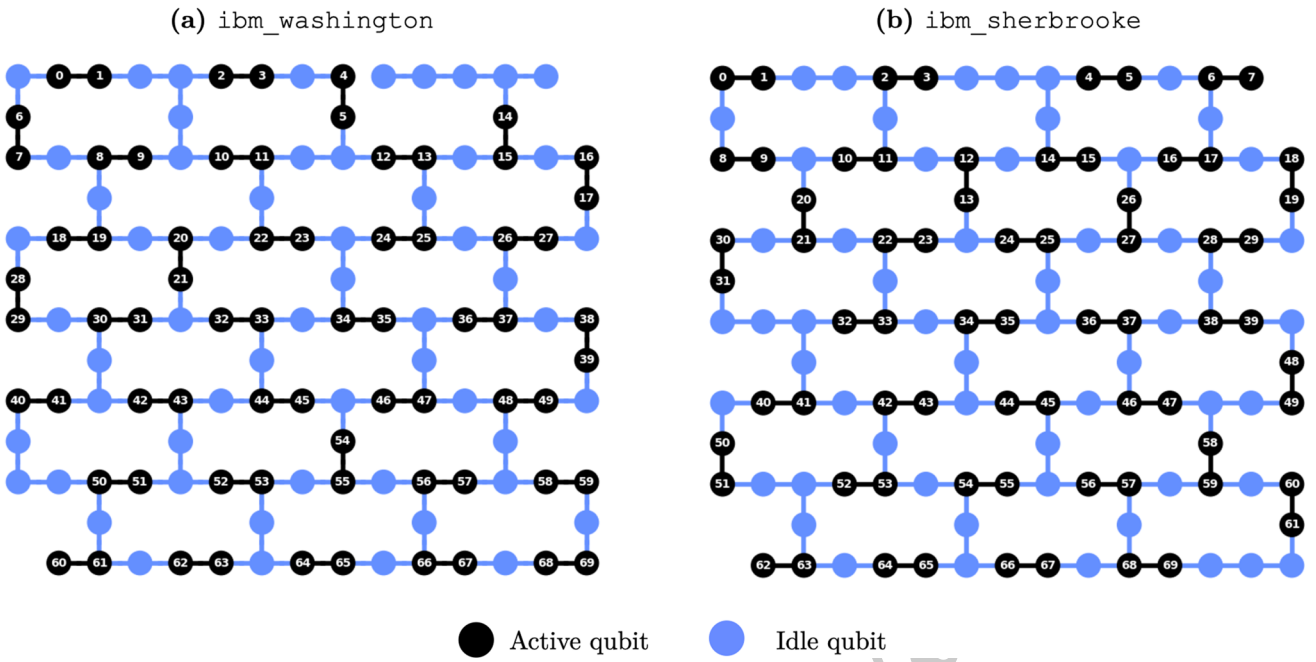

Fig. 6 The mapping between logical and physical qubits for the QMP circuits used in our experiments on **a** `ibm_washington` (`Eagle r1` processor) and **b** `ibm_sherbrooke` (`Eagle r3` processor). Both machines have 127 physical qubits and have a similar (but not identical) qubit connectivity graph. `ibm_washington` has a QV of 64 while `ibm_sherbrooke` has a QV of 32. On `ibm_washington`, the native gates are CX, ID, RZ, SX, and X and `ibm_sherbrooke` uses the same single qubit gates but ECR gates are used for 2-qubit interactions. Idle qubits are shown as blue circles and active qubits are black circles containing integers indexing the logical qubits

Finally, we mention partial measurement in QMP. After the whole measurement of a QMP circuit, we need to extract the measurements from each parallelized circuit. This is called partial measurement and is implemented by post-processing after the whole measurement since the partial measurement probability is the sum of all other uninvolved qubits measurement probability. This is detailed in Park et al. (2023) and in the Supplementary Information. Since the example code in Appendix A in Park et al. (2023) iterates over the 2^n where n is the number of the measured qubits, the computing time increases exponentially as more qubits are measured. Hence, we re-implement the partial measurement code, which extracts the partial measurement out of the whole measurement, to depend on the number of shots regardless of the number of the measured qubits. Our new implementation is given in the Supplementary Information.

5.3 QMP experiments with the gun-point data set

We conduct QMP experiments using the gun-point data set described in Section 4.2 where the time dimension is decimated by 3× (p is reduced from 150 to 50) to reduce the total QPU run time to a reasonable level. An optimal simulated (ideal) model for the decimated data set is first obtained using the same method described in Section 4.2. Using the $\boldsymbol{\theta}^\star$ derived from the simulated model, we generate the quantum circuits used to calculate the matrix elements of the kernel matrices required to train an SVM. These circuits are deployed on two 127-qubit superconducting QPUs: `ibm_washington` (64 QV) and `ibm_sherbrooke` (32 QV). The experiments utilize the maximum number of active qubits which can be used in a single QMP run. For both devices, including the buffer qubit constraint, this number is 70. Figure 6 shows the qubit layout on `ibm_washington` (Fig. 6a) and `ibm_sherbrooke` (Fig. 6b). In addition to mapping considerations already discussed in Section 5.2, we manually choose the qubits with low error ratios (read assignment error, etc.). Considering each circuit has two qubits, we can run 35 circuits in parallel with a single QMP run. The Trial Reduction Factor (TRF) (Das et al. 2019) describes the ratio of the number of trials (shots) that executes individually (baseline) to the number of trials (shots) of the QMP circuit. This factor represents the efficiency of the QMP in terms of trials (shots). In our case, the TRF is 35. The QMP used 8192 shot numbers to measure 35 circuits, whereas non-QMP implementations need 8192 × 35 shots to measure 35 circuits.

Recalling the total size of the training and testing data sets in the decimated gun-point data set, a naive serial QPU implementation (without QMP) requires 436, 250 individual calls to a quantum device to calculate all of the relevant kernel matrices to train and test an SVM. By applying QMP, the

total number of calls to the QPU is *significantly* reduced by a factor of 35 to 12, 465 because the QMP circuit parallelizes 35 quantum circuits. As a preliminary step, we first verified that our QMP implementation produced consistent outputs with serial execution by comparing the output of both using a quantum circuit simulator. The outputs for serial and parallel are in perfect agreement. The details of these tests, and others, are given in the Supplementary Information. Importantly, in practice, QMP reduces the queuing time of the circuit. Because of the much-reduced number of calls to the QPU, there are far fewer circuits in the queuing system of an individual device. The advantage of this is two-fold as (i) the wait time is reduced for the user and other users and (ii) schedulers often use a "fair-share" allowance on the number of jobs submitted by a given user. The fewer number of circuits sent means that job priority levels administered by the scheduler remain high. Finally, we note that because of our choice to use two separate 127 qubit machines, circuits were sent to both in parallel, thus two queues were being occupied at all times which itself leads to a practical speedup.

The results of our QMP experiments are shown in Fig. 7. Figure 7a compares F_1, A_B, and RA scores for the simulated results (red bars) and the QMP runs executed on `ibm_washington` and `ibm_sherbrooke` (QPU $\times 2$; grey bars). Compared with the simulator results from Section 4.2, we see that MKL SVM on decimated gun-point data set gives rise to lower scores for all three metrics. This is intuitive as the algorithm is being shown fewer data points (100 fewer time points) in training. When run on 2× QPU, scores fall $\approx$ 10–20% across the board as a result of all of the various types of hardware noise. We should note that besides the use of buffer qubits and selection of low-error qubits on the machines, no other optimizations/error mitigation techniques were used so the results presented in Fig. 7 can be considered a baseline. In reality, more highly optimized transpiler passes could be used alongside a whole host of dedicated error mitigation techniques including (but not limited to) Pauli twirling, dynamical decoupling, and various flavors of read-out mitigation. Figure 7b shows the error in the kernel weights $\boldsymbol{\eta}$ as a function of time (i.e., η_t is shown). Errors exceeding 25% occur at low and high t, while errors are very small at intermediate t. Interestingly (although not shown), areas where the weights themselves are larger (at intermediate t) have the smallest errors which means despite a significant amount of noise entering our kernel matrix computations, the kernel weight values are quite robust. Figure 7c and d show the SVM decision function $\mathcal{D}^\star(\boldsymbol{x})$ (discussed in Section 2.3) evaluated using all 150 of the testing time-series instances using where the TSHK was either computed with an ideal quantum circuit simulator (Fig. 7c) or calculated on real devices using QMP (Fig. 7d). Recall that $\text{sgn}[\mathcal{D}^\star(\boldsymbol{x})]$ distinguishes the predicted binary class label such that points above the black dotted line at $\mathcal{D}^\star(\boldsymbol{x}) = 0$ are predicted as class 1, while those below it are predicted as class -1. Because each point is colored by its truth label y_i, it is possible to see from Fig. 7c and 7d which points are correctly and incorrectly classified. Looking first at the simulator results (Fig. 7c), it can be seen that many points sit *very close* to $\mathcal{D}^\star(\boldsymbol{x}) = 0$ meaning their classification is a *close call*. This finding is important as it provides some insight into why performance metrics decrease for the QPU results. That is, we can see from Fig. 7d that those *close call* points which were sitting close to $\mathcal{D}^\star(\boldsymbol{x}) = 0$ for the simulated results have been smeared across the decision boundary by a non-trivial propagation of error incurred by (i) training the SVM with a noisy (and only approximately positive semi-definite) Gram matrix and (ii) making predictions using already noisy trained SVM model using a similarly noisy kernel function. We observe also that those points which were sitting further from the decision boundary in the simulator results now even further from it.

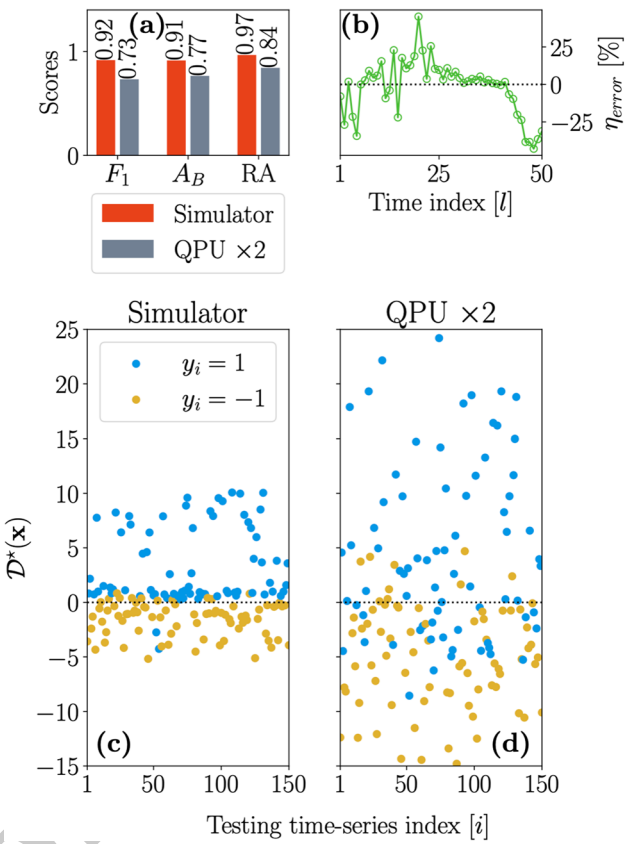

Fig. 7 Simulator versus device results on the sub-sampled gun-point data set. **a** The F_1, A_B (balanced accuracy), and RA (ROC AUC) scores for the simulator (red) and device (grey; QPU $\times 2$) results. **b** The percentage error in the kernel weights from the device results vs the simulator given as $\eta_{error} = 100 \cdot (\eta_{device} - \eta_{ideal})/\eta_{ideal}$ plotted as a function of time index l. **c**, **d** The SVM decision function $\mathcal{D}^\star(\boldsymbol{x})$ for each of the 150 testing time-series using the simulated kernel matrices (**c**) or the device kernel matrices (**d**). Points are colored by their truth label. Blue points above the black dotted line are correctly classified, while those below it are incorrectly classified. The reverse is true for gold points

Following the approach taken in Hubregtsen et al. (2022), errors incurred by the processes described in points (i) and (ii) above can be mitigated against. Using the Tikhonov regularization scheme (Hubregtsen et al. 2022), Gram matrices K can regularized (be made positive semi-definite) with a simple correction:

$$K^{\text{reg}} = \begin{cases} K - \epsilon_{\min} I & \text{if } \epsilon_{\min} < 0 \\ K & \text{otherwise} \end{cases} \tag{18}$$

for regularized gram matrix K^{reg}, smallest eigenvalue in the spectrum of K, $\epsilon_{\min}$ and identity matrix I. We apply the correction of Eq. 18 to each of the noisy time-dependent Gram matrices K_t and use the regularized output to find the optimal kernel weights $\eta_t^\star$ given in Eq. 11. Finally, we build a regularized combined kernel using Eq. 6. We now use the regularized combined kernel as input to a SVM to classify the gun-point data set. We find that all considered metrics are improved: $A_B = 0.81$, $F_1 = 0.82$, and $RA = 0.90$. This shows the success of even simple regularization techniques and supports the need for future works assessing the best possible way to use kernel methods on noisy QPUs.

6 Conclusions and perspectives

In this work, we proposed and demonstrated a hybrid quantum-classical supervised ML algorithm for time-series classification. The algorithm works by utilizing time-dependent inner product spaces as generated by a common time-evolution operator and combining the inner products using classical MKL techniques. Together, the components of the algorithm create a temporally aware QCC-net able to tailor kernel functions for arbitrary time-series data. When these tailored kernel functions were used with kernelized SVM, we found classification performance to be comparable with purely classical methods for the well-known gun-point data set, although a great deal of the performance stemmed from the classical portions of the algorithm. Through conducting experiments on the synthetic `moons2circles` data set, we verified that the TSHK is most effective in situations where the correct notion of similarity between pairs of data points evolves with time and there exists a shared time axis in the data as was described in Section 2.1.

Furthermore, we developed a method to study the resulting time-dependence present within a trained time-series kernel function. We must stress that algorithms trained to explicitly learn time-dependent inner product spaces are novel even within the classical ML literature, and indeed, we do not know of an existing classical approach that can precisely emulate the method proposed in this work.

We also found that the total run time of our method can be greatly reduced by utilizing the state-of-the-art method, QMP. That is, because the computation of quantum kernel matrix elements is an *embarrassingly parallel* problem, many matrix elements can be computed in parallel by distributing multiple quantum threads across large QPUs. We demonstrated this parallelism using two 127-qubit superconducting quantum processors to perform SVM classification on a decimated version of the gun-point data set. With this method, we were able to reduce the total number of required QPU calls by 35×, which, assuming constant job queuing times for both 127-qubit QPUs, leads to a 70× speed-up compared to serial execution on a single QPU. We have therefore demonstrated, for the first time, that QMP is a valuable tool for significantly speeding up QML algorithms on present and near future NISQ machines. With recent advancements in low-latency CPU-QPU interactions by `Qiskit Runtime` and others, in the near future, we expect to see the emergence of real-time training of error-mitigated QML where the quantum components of the algorithm are accelerated by QMP and the classical parts are accelerated using conventional parallel processing paradigms like MPI and OpenMP. This work, therefore, makes strides along the path toward the much-anticipated intersection of quantum computation and classical high performance computing.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s42484-024-00149-0.

Acknowledgements We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum team. This research used quantum computing resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award DDR-ERCAP0024165.

Author Contributions Jack S. Baker made the figures, conducted the numerical experiments on quantum circuit simulators, wrote the manuscript, and helped develop the theoretical method. Gilchan Park conducted the QMP experiments and contributed to the manuscript. Kwangmin Yu oversaw the QML experiments and contributed to the manuscript and the supplemental information. Ara Ghukasayan helped to develop the theoretical method and reviewed the manuscript. Oktay Goktas oversaw the numerical experiments and reviewed the manuscript. Santosh Kumar Radha seeded the idea of using time evolution operators to create a time dependent quantum kernel function and reviewed the manuscript.

Funding This work was partially supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, Grant No. DE-SC-0012704 (GP, KY) and the SBU-BNL Seed Grant 2019, 2023 (KY).

Data and code availability The data used to produce Figs. 3, 4, and 7 can be found at https://doi.org/10.5281/zenodo.7996534 alongside a code example implementing a QCC-net for the Sine vs Cosine classification shown in Fig. 1.

Declarations

Conflict of interest The authors declare no competing interests.

References

Agrawal A, Amos B, Barratt S, Boyd S, Diamond S, Kolter JZ (2019a) Differentiable convex optimization layers. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems, vol 32. Curran Associates, Inc

Agrawal A, Barratt S, Boyd S, Busseti E, Moursi WM (2019b) Differentiating through a cone program. https://doi.org/10.48550/ARXIV.1904.09043

Aiolli F, Donini M (2015) EasyMKL: a scalable multiple kernel learning algorithm. Neurocomput 169:215. https://doi.org/10.1016/j.neucom.2014.11.078

Aiolli F, Martino GDS, Sperduti A (2008) A kernel method for the optimization of the margin distribution. In: Artificial neural networks - ICANN 2008. Springer Berlin Heidelberg, pp 305–314. https://doi.org/10.1007/978-3-540-87536-9_32

Badiane M, O'Reilly M, Cunningham P (2018) Kernel methods for time series classification and regression. In: AICS pp 54–65

Bailly A (2018) Time series classification algorithms with applications in remote sensing. Ph.D. thesis, Université Rennes 2

Baker JS, Horowitz H, Radha SK, Fernandes S, Jones C, Noorani N, Skavysh V, Lamontangne P, Sanders BC (2022) Quantum variational rewinding for time series anomaly detection. https://doi.org/10.48550/ARXIV.2210.16438

Bergholm V, Izaac J, Schuld M, Gogolin C, Ahmed S, Ajith V, Alam MS, Alonso-Linaje G, AkashNarayanan B, Asadi A, Arrazola J. M, Azad U, Banning S, Blank C, Bromley TR, Cordier BA, Ceroni J, Delgado A, Di Matteo O, Dusko A, Garg T, Guala D, Hayes A, Hill R, Ijaz A, Isacsson T, Ittah D, Jahangiri S, Jain P, Jiang E, Khandelwal A, Kottmann K, Lang R. A, Lee C, Loke T, Lowe A, McKiernan K, Meyer JJ, Montañez-Barrera JA, Moyard R, Niu Z, O'Riordan LJ, Oud S, Panigrahi A, Park C-Y, Polatajko D, Quesada N, Roberts C, Sá N, Schoch I, Shi B, Shu S, Sim S, Singh A, Strandberg I, Soni J, Száva A, Thabet S, Vargas-Hernández RA, Vincent T, Vitucci N, Weber M, Wierichs D, Wiersema R, Willmann M, Wong V, Zhang S, Killoran N (2018) Pennylane: automatic differentiation of hybrid quantum-classical computations. https://doi.org/10.48550/ARXIV.1811.04968

Bharti K, Cervera-Lierta A, Kyaw TH, Haug T, Alperin-Lea S, Anand A, Degroote M, Heimonen H, Kottmann JS, Menke T, Mok W-K, Sim S, Kwek L-C, Aspuru-Guzik A (2022) Noisy intermediate-scale quantum algorithms. Rev Mod Phys 94:015004. https://doi.org/10.1103/RevModPhys.94.015004

Bjorken JD, Drell SD (1965) Relativistic quantum fields. McGraw-Hill

Blázquez-García A, Conde A, Mori U, Lozano JA (2021) A review on outlier/anomaly detection in time series data. ACM Comput Surv 54:1. https://doi.org/10.1145/3444690

Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Rev 60:223. https://doi.org/10.1137/16m1080173

Bowles J, Wright VJ, Farkas M, Killoran N, Schuld M (2023) Contextuality and inductive bias in quantum machine learning. arXiv:2302.01365

Caro MC, Huang H-Y, Ezzell N, Gibbs J, Sornborger AT, Cincio L, Coles PJ, Holmes Z (2022) Out-of-distribution generalization for learning quantum dynamics. https://doi.org/10.48550/ARXIV.2204.10268

Choi K, Yi J, Park C, Yoon S (2021) Deep learning for anomaly detection in time-series data: review, analysis, and guidelines. IEEE Access 9:120043. https://doi.org/10.1109/access.2021.3107975

Cîrstoiu C, Holmes Z, Iosue J, Cincio L, Coles P. J, Sornborger A (2020) Variational fast forwarding for quantum simulation beyond the coherence time. npj Quantum Inf 6. https://doi.org/10.1038/s41534-020-00302-0

Clark S, Hyndman RJ, Pagendam D, Ryan LM (2020) Modern strategies for time series regression. Int Stat Rev 88. https://doi.org/10.1111/insr.12432

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273

Cross AW, Bishop LS, Sheldon S, Nation PD, Gambetta JM (2019) Validating quantum computers using randomized model circuits. Phys Rev A 100:032328

Das P, Tannu SS, Nair PJ, Qureshi M (2019) A case for multi-programming quantum computers. In: Proceedings of the 52nd Annual IEEE/ACM international symposium on microarchitecture. pp 291–303

Deb C, Zhang F, Yang J, Lee SE, Shah KW (2017) A review on time series forecasting techniques for building energy consumption. Renewa Sustain Energy Rev 74:902. https://doi.org/10.1016/j.rser.2017.02.085

Fábregues de los Santos L (2017) Forecasting financial time series using multiple Kernel Learning. Master's thesis, Universitat Politècnica de Catalunya

Farhi E, Goldstone J, Gutmann S (2014) A quantum approximate optimization algorithm. arXiv:1411.4028

Fawaz HI, Forestier G, Weber J, Idoumghar L, Muller P-A (2019) Deep learning for time series classification: a review. Data Min Knowl Discov 33:917. https://doi.org/10.1007/s10618-019-00619-1

Ghukasyan A, Baker JS, Goktas O, Carrasquilla J, Radha S. K (2023) Quantum-classical multiple kernel learning. arXiv:2305.17707 [quant-ph]

Gibbs J, Holmes Z, Caro MC, Ezzell N, Huang H-Y, Cincio L, Sornborger AT, Coles PJ (2022) Dynamical simulation via quantum machine learning with provable generalization. https://doi.org/10.48550/ARXIV.2204.10269

Gönen M, Alpaydın E (2011) Multiple kernel learning algorithms. The J Mach Learn Res 12:2211

Gönen M, Alpaydın E (2011) Multiple kernel learning algorithms. The J Mach Learn Res 12:2211

Hadfield S, Wang Z, O'Gorman B, Rieffel EG, Venturelli D, Biswas R (2019) From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algoritm 12

Hastie T, Tibshirani R, Friedman JH, Friedman JH (2009) The elements of statistical learning: data mining, inference, and prediction. vol 2. Springer, p 134

Havlíček V, Córcoles AD, Temme K, Harrow AW, Kandala A, Chow JM, Gambetta JM (2019) Supervised learning with quantum-enhanced feature spaces. Nat 567:209. https://doi.org/10.1038/s41586-019-0980-2

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735. https://doi.org/10.1162/neco.1997.9.8.1735

Horowitz H, Rao P, Radha SK (2022) A quantum generative model for multi-dimensional time series using Hamiltonian learning. https://doi.org/10.48550/ARXIV.2204.06150

Hubregtsen T, Wierichs D, Gil-Fuster E, Derks P-JHS, Faehrmann PK, Meyer JJ (2022) Training quantum embedding kernels on near-

term quantum computers. Phys Rev A 106:042431. https://doi.org/10.1103/PhysRevA.106.042431
Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP (2017) On large-batch training for deep learning: generalization gap and sharp minima. arXiv:1609.04836 [cs.LG]
Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. arXiv:1412.6980
Kübler JM, Buchholz S, Schölkopf B (2021) The inductive bias of quantum kernels. arXiv:2106.03747
Liu L, Dou X (2021) QuCloud: a new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In: 2021 IEEE International symposium on High-Performance Computer Architecture (HPCA). IEEE, pp 167–178
Lloyd S, Schuld M, Ijaz A, Izaac J, Killoran N (2020) Quantum embeddings for machine learning. arXiv:2001.03622
McClean JR, Boixo S, Smelyanskiy VN, Babbush R, Neven H (2018) Barren plateaus in quantum neural network training landscapes. Nat Commun 9. https://doi.org/10.1038/s41467-018-07090-4
Melis G, Kočiský T, Blunsom P (2019) Mogrifier lstm. https://doi.org/10.48550/ARXIV.1909.01792
Mitarai K, Negoro M, Kitagawa M, Fujii K (2018) Quantum circuit learning. Phys Rev A 98. https://doi.org/10.1103/physreva.98.032309
Murali P, McKay DC, Martonosi M, Javadi-Abhari A (2020) Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In: Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems. pp 1001–1016
Nguyen T, Baraniuk R, Bertozzi A, Osher S, Wang B (2020) Momentumrnn: integrating momentum into recurrent neural networks. In: Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H (eds) Advances in neural information processing systems, vol 33. Curran Associates, Inc., pp 1924–1936
Niu S, Todri-Sanial A (2021) Enabling multi-programming mechanism for quantum computing in the NISQ era. arXiv:2102.05321
Niu S, Todri-Sanial A (2022) How parallel circuit execution can be useful for NISQ computing? In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, pp 1065–1070
O'Donoghue B (2021) Operator splitting for a homogeneous embedding of the linear complementarity problem. SIAM J Optim 31:1999
O'Donoghue B, Chu E, Parikh N, Boyd S (2016) Conic optimization via operator splitting and homogeneous self-dual embedding. J Optim Theor Appl 169:1042. http://stanford.edu/~boyd/papers/scs.html
O'Donoghue B, Chu E, Parikh N, Boyd S (2016a) Conic optimization via operator splitting and homogeneous self-dual embedding. J Optim Theor Appl 169:1042. http://stanford.edu/~boyd/papers/scs.html
Ohkura Y (2021) Crosstalk-aware NISQ Multi-programming. Keio Univ, Tokyo, Japan, Fac Policy Manag
Ohkura Y, Satoh T, Van Meter R (2022) Simultaneous execution of quantum circuits on current and near-future NISQ systems. IEEE Trans Quantum Eng
Park G, Zhang K, Yu K, Korepin V (2023) Quantum multi-programming for Grover's search. Quantum Inf Process 22:54
Pesah A, Cerezo M, Wang S, Volkoff T, Sornborger AT, Coles PJ (2021) Absence of barren plateaus in quantum convolutional neural networks. Phys Rev X 11:041011. https://doi.org/10.1103/PhysRevX.11.041011
Radha SK (2021) Quantum constraint learning for quantum approximate optimization algorithm. https://doi.org/10.48550/ARXIV.2105.06770
Ratanamahatana CA, Keogh E (2005) Three myths about dynamic time warping data mining. In: Proceedings of the 2005 SIAM international conference on data mining. SIAM, pp 506–510
Rüping S (2001) SVM kernels for time series analysis. Tech. Rep. (Technical report 2001)
Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust, Speech, Signal Process 26:43. https://doi.org/10.1109/TASSP.1978.1163055
Schuld M, Bocharov A, Svore KM, Wiebe N (2020) Circuit-centric quantum classifiers. Phys Rev A 101:032308. https://doi.org/10.1103/PhysRevA.101.032308
Schuld M, Bergholm V, Gogolin C, Izaac J, Killoran N (2019) Evaluating analytic gradients on quantum hardware. Phys Rev A 99. https://doi.org/10.1103/physreva.99.032331
Schuld M, Killoran N (2019) Quantum machine learning in feature Hilbert spaces. Phys Rev Lett 122. https://doi.org/10.1103/physrevlett.122.040504
Stone MH (1932) On one-parameter unitary groups in Hilbert space. The Annal Math 33:643. https://doi.org/10.2307/1968538
Tino P (2020) Dynamical systems as temporal feature spaces. The J Mach Learn Res 21:1649
Torres JF, Hadjout D, Sebaa A, Martínez-Álvarez F, Troncoso A (2021) Deep learning for time series forecasting: a survey. Big Data 9:3. https://doi.org/10.1089/big.2020.0159
Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L u, Polosukhin I (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. Curran Associates, Inc
Welch J, Greenbaum D, Mostame S, Aspuru-Guzik A (2014) Efficient quantum circuits for diagonal unitaries without ancillas. New J Phys 16:033040. https://doi.org/10.1088/1367-2630/16/3/033040
Wiebe N, Berry D, Høyer P, Sanders BC (2010) Higher order decompositions of ordered operator exponentials. J Phys A: Math Theor 43:065203. https://doi.org/10.1088/1751-8113/43/6/065203
Yang C-HH, Tsai Y-Y, Chen P-Y (2021) Voice2series: reprogramming acoustic models for time series classification. In: International conference on machine learning. PMLR, pp 11808–11819
Zerveas G, Jayaraman S, Patel D, Bhamidipaty A, Eickhoff C (2021) A transformer-based framework for multivariate time series representation learning. In: Proceedings of the 27th ACM SIGKDD conference on knowledge discovery; data mining. ACM, https://doi.org/10.1145/3447548.3467401
Zhang C, Kuppannagari SR, Kannan R, Prasanna VK (2008) Generative adversarial network for synthetic time series data generation in smart grids. In: 2018 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm). IEEE, https://doi.org/10.1109/smartgridcomm.2018.8587464