Kernel Approximation on a Quantum Annealer for Remote Sensing Regression Tasks

Edoardo Pasetto, Morris Riedel, *Member, IEEE*, Kristel Michielsen, and Gabriele Cavallaro, *Senior Member, IEEE*

***Abstract*—The increased development of quantum computing hardware in recent years has led to increased interest in its application to various areas. Finding effective ways to apply this technology to real-world use-cases is a current area of research in the remote sensing community. This article proposes an adiabatic quantum kitchen sinks (AQKS) kernel approximation algorithm with parallel quantum annealing on the D-Wave Advantage quantum annealer. The proposed implementation is applied to support vector regression and Gaussian process regression algorithms. To evaluate its performance, a regression problem related to estimating chlorophyll concentration in water is considered. The proposed algorithm was tested on two real-world datasets and its results were compared with those obtained by a classical implementation of kernel-based algorithms and a random kitchen sinks implementation. On average, the parallel AQKS achieved comparable results to the benchmark methods, indicating its potential for future applications.**

***Index Terms*—Parallel quantum annealing, quantum annealing (QA), quantum computing (QC), regression, remote sensing (RS).**

I. Introduction

THE task of estimating biophysical quantities from remote sensing (RS) measurement data is a well-studied problem in the research community, covering a range of applications such as water chlorophyll concentration estimation [1], [2], [3], ozone concentration estimation [4], and crop yield prediction [5]. The task can be interpreted as an inverse modeling problem whose objective is to find a relationship between acquired measurements of some specific physical quantities and a value of interest [1]. On a formal point of view the objective is to determine a function $y = f(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$, where $\mathbf{x} \in \mathbb{R}^d$ is the input feature vector containing the data of the optical measurements and the scalar $y \in \mathbb{R}$ is the quantity of interest to be determined. The learning of process of the function $f(.)$ is carried out by observing a training set of data observation, i.e a set of N pairs of observation measurements vectors and their corresponding target value $\{(\mathbf{x}_i, y_i),\ i = 1, \ldots, N\}$. Regression tasks in remote sensing (RS) have been studied by applying different supervised learning algorithms and among the most popular are support vector regression (SVR) [6], [7], kernel ridge regression (KRR) [8], and Gaussian process regression (GPR) [9]. A common feature of these methods is the usage of a *kernel function* $k(\mathbf{x}, \mathbf{x}')$, which allows to calculate the dot product between a nonlinear map of the input vectors in a transformed feature space taking as argument the original input vectors, i.e., $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$, where $\phi(.)$ is a nonlinear feature map. One of the advantages of using kernel methods comes from the so-called *kernel trick:* if in the mathematical formulation of a learning algorithm feature vectors appear only as dot products between them, it is possible to "kernelize" the algorithm by substituting such products with the kernel function calculated on the same feature vectors [10], [11]. The main characteristic of this procedure is that it is not necessary to know the nonlinear feature mapping $\phi(.)$ nor the transformed vectors themselves since the only information needed can be obtained implicitly by the evaluation of the kernel function. Kernel methods, however, tend to scale badly as the size of the training set increases [12]. Starting from this observation, Rahimi et al. [12], [13] proposed the random kitchen sinks (RKS) kernel approximation algorithm, which approximates the kernel function by using randomized features. This procedure, also known as Random Fourier Features, therefore does not employ a kernel function but instead explicitly generates transformed feature vectors through randomization.

Quantum computing (QC) [14], [15] is a computational model based upon the properties of quantum mechanics that was theoretically proven to have the potential to outperform classical computers in terms of computational complexity on some specific tasks [16], [17]. However, the availability of a reliable large-scale quantum computer might still be a distant goal [18]. The growing interest towards the application of different QC algorithms to enhance machine learning (ML) frameworks laid the foundations for the development of the research field of quantum machine learning (QML) [19], [20], [21], [22], [23]. In the context of RS, QML have been applied to image classification

Manuscript received 31 January 2023; revised 27 August 2023; accepted 31 December 2023. Date of publication 5 January 2024; date of current version 19 January 2024. This work was supported in part by the project JUNIQ that has received funding from the German Federal Ministry of Education and Research (BMBF) and the Ministry of Culture and Science of the State of North Rhine-Westphalia, in part by the European High-Performance Computing Joint Undertaking (JU) under Grant EUROCC2 project, in part by EU/EEA states under Grant 101101903, in part by the Center of Excellence (CoE) Research on AI-, and in part by Simulation-Based Engineering at Exascale (RAISE) receiving funding from EU's Horizon 2020 Research and Innovation Framework Programme under Grant H2020-INFRAEDI-2019-1 and Grant 951733. *(Corresponding author: Gabriele Cavallaro.)*

Edoardo Pasetto and Kristel Michielsen are with Jülich Supercomputing Centre, Wilhelm-Johnen Straße, 52428 Jülich, Germany, also with RWTH Aachen University, D-52056 Aachen, Germany, and also with AIDAS, 52425 Jülich, Germany (e-mail: e.pasetto@fz-juelich.de; k.michielsen@fz-juelich.de).

Morris Riedel and Gabriele Cavallaro are with the University of Iceland, 107 Reykjavik, Iceland, also with Jülich Supercomputing Centre, Wilhelm-Johnen Straße, 52428 Jülich, Germany, and also with AIDAS, 52425 Jülich, Germany (e-mail: morris@hi.is; g.cavallaro@fz-juelich.de).

Digital Object Identifier 10.1109/JSTARS.2024.3350385

through the usage of a hybrid quantum-classical neural network whose quantum layer was implemented with a parametrized quantum circuit [24], [25], [26].

A QML-based implementation of the Random Fourier features has been recently proposed with gate-based quantum computing [27] and quantum annealing (QA) [28]. In the QA-based implementation, also referred to as adiabatic quantum kitchen sinks (AQKS), data are linearly encoded in the Hamiltonian of a quantum system which is then evolved, and the measurement value taken at the end of the process is then used to generate the transformed feature vectors that are then used to train a support vector machine (SVM) for binary classification tasks. In this work, the AQKS kernel approximation algorithm is applied to two different kernel-based regression algorithms: SVR and GPR on two real RS datasets related to chlorophyll concentration estimation. The obtained results are then compared with those obtained by the corresponding traditional kernel-based versions and those obtained by the same algorithm trained using the classical RKS kernel approximation algorithm. The implementation of the AQKS kernel approximation algorithm is done using a D-Wave Advantage system quantum annealer, whereas the work in [28] simulated the quantum system through trotterization. Moreover, since the workflow of AQKS requires to solve with the quantum annealer many problems of small size, the concept of *parallel quantum annealing* [29] was used in order to reduce the computational time during the learning process by running multiple problem instances in the same annealing cycle. This is possible because, if two or more problems are independent, they can be solved in the same annealing cycle by solving the optimization problem obtained by summing them together. For the sake of clarity in the notation, the algorithms implemented with a traditional kernel, the AQKS kernel approximation, and the RKS kernel approximation are referred to as *classical*, *quantum*, and *RKS-based*, respectively. Our contributions in this work can be summarized as follows: Implementation of AQKS on a real QA device, application of such a scheme to regression problem with two different algorithms, integration of AQKS with parallel QA to reduce the computational time, and to the best of our knowledge, first time application of such a scheme to a real RS use-case.

II. Quantum Annealing and QUBO Problem Formulation

To solve a problem with a quantum annealer it is necessary to reformulate it as as a quadratic binary unconstrained optimization (QUBO), which corresponds to the optimization of the following energy function:

$$\min_{a_1,\ldots,a_N} \sum_{i=1}^{N} \sum_{j=i+1}^{N} a_i Q_{ij} a_j \tag{1}$$

where $a_i \in \{0,1\}$ and Q is an upper-triangular matrix containing the coefficients of the problem that is referred to as QUBO weight matrix. By defining $\mathbf{a} \in \{0,1\}^N \triangleq [a_1, \ldots a_N]$ it is possible to rewrite (1) in matrix product form as

$$\min_{\mathbf{a}} \mathbf{a}^T Q \mathbf{a}. \tag{2}$$

Alternatively, it is also possible to reformulate the problem as a Ising spin model [30], which is a binary model whose variables take value in the set $\{-1,-1\}$. For QA purposes both problem formulations can be used.

III. Kernel Regression Methods

In this section a description of the classical kernel-based regression methods is now provided. In principle any symmetric and positive semidefinite function $k(\mathbf{x},\mathbf{x}')$ can be used as kernel function [12]. In ML, one of the most popular choice for kernel function is the radial basis function (RBF) kernel, which has the property of depending only on the distance of the inputs, i.e: $k(\mathbf{x},\mathbf{x}') = k(||\mathbf{x}-\mathbf{x}'||)$. The formula of the RBF is as follows:

$$k(\mathbf{x},\mathbf{x}') = exp\left(\frac{||\mathbf{x}-\mathbf{x}'||}{\gamma}\right). \tag{3}$$

The prediction function of the kernel-based algorithms used in this work can be formulated as a weighted sum of kernel function evaluations between the N training data points and the input vector $\mathbf{x}$

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i,\mathbf{x}) + b \tag{4}$$

where $\alpha_1,\ldots,\alpha_N$ are a set of scalar whose value is determined in the learning phase on the training set. The prediction function is linear with respect to the kernel function evaluations so the nonlinear modeling in the original feature space is achieved by applying a linear model in the transformed feature space. In the following, it will be denoted as $\mathbf{X}$ the $N \times d$ *design matrix* in which each of its row corresponds to a training sample, i.e., $\mathbf{X}[i,:] = \mathbf{x}_i \; i = 1,\ldots,N$ and as $\mathbf{y} \in \mathbb{R}^N$ the corresponding target vector. Let us also define as $\mathbf{K}$ the $N \times N$ symmetric matrix, referred to as *Gram* matrix, that stores the kernel function evaluation between every pair of training sample $\mathbf{x}_i$ and $\mathbf{x}_j$, i.e., $\mathbf{K}_{ij} = \mathbf{K}_{ji} = k(\mathbf{x}_i,\mathbf{x}_j)$.

A. Support Vector Regression

The formulation of the SVR can be obtained by considering the optimization of a regularized regression problem where the considered loss function is a $\epsilon-$ insensitive loss function [31], i.e., a function that gives an error only if the absolute difference between the actual value and the predicted one is greater than a value $\epsilon > 0$ [10]

$$\mathcal{L}_\epsilon(f(\mathbf{x})-y) = \begin{cases} 0, & \text{if } |f(\mathbf{x})-y| < \epsilon; \\ |f(\mathbf{x})-y|, & \text{otherwise.} \end{cases} \tag{5}$$

The loss function to be minimized is then

$$C\sum_{n=1}^{N} \mathcal{L}_\epsilon(f(\mathbf{x}_n)-y_n) + \frac{1}{2}||\mathbf{w}||^2. \tag{6}$$

In the formula, C is a parameter that controls the overfitting that by convention multiplies the error term in the equation, and therefore can be thought as a (inverse)-regularization parameter [10]. The vector $\mathbf{w}$ is associated with the linear coefficients in the transformed feature space.

It can be shown that the training of the SVR amounts to the solving of the following constrained optimization problem [10]:

$$L(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n - \hat{\alpha}_n)(\alpha_m - \hat{\alpha}_m)k(\mathbf{x}_n, \mathbf{x}_m) + - \epsilon\sum_{n=1}^{N}(\alpha_n + \hat{\alpha}_n) + \sum_{n=1}^{N}(\alpha_n - \hat{\alpha}_n)y_n \tag{7}$$

subject to the constraints

$$\sum_{n=1}^{N}(\alpha_n - \hat{\alpha}_n) = 0 \tag{8a}$$

$$0 \leq \alpha_n \leq C \tag{8b}$$

$$0 \leq \hat{\alpha}_n \leq C \tag{8c}$$

with respect to the variables α_i and $\hat{\alpha}_i$ with $i \in 1, \ldots, N$. Once the values of $\alpha_1, \ldots, \alpha_N$ and $\hat{\alpha}_1, \ldots, \hat{\alpha}_N$ have been determined a prediction on an input sample $\mathbf{x}$ can then be made through the formula

$$f(\mathbf{x}) = \sum_{n=1}^{N}(\alpha_n - \hat{\alpha}_n)k(\mathbf{x}, \mathbf{x}_n) + b. \tag{9}$$

The value of b can be obtained from any point for which $0 < \alpha_n < C$ or $0 < \hat{\alpha_n} < C$ through the formula

$$b = t_n - \epsilon - \sum_{m=1}^{N}(\alpha_m - \hat{\alpha}_m)k(\mathbf{x}_n, \mathbf{x}_m). \tag{10}$$

It is preferable, however, to average over multiple data points in order to get a more stable estimation [10].

B. Gaussian Process Regression

The regression approach of GPR is different from that of SVR because it provides a output distribution of the target y instead of a point estimation. Such probability distribution is Gaussian and therefore, it is completely determined by the value of the mean μ^* and variance σ^*. In GPR, the relationship between the input vectors stored in $\mathbf{X}$ and the target values is modeled as a sum between a Gaussian multivariate function $\mathcal{N}(\mathbf{0}, \mathbf{K})$ and a independent noise component $\mathcal{N}(\mathbf{0}, \beta^{-1}\mathbf{I}_N)$. The Gram matrix is used to construct the covariance matrix that is used to model the generation process of the training set. By the properties of the Gaussian function [10] the target values assume the following probability distribution:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \beta\mathbf{I}_N). \tag{11}$$

To make a prediction on a unseen input $\mathbf{x}$, let us consider $\mathbf{X}^*$ the $N+1 \times d$ matrix obtained by vertically concatenating the vector $\mathbf{x}$ to the matrix $\mathbf{X}$, i.e., the last row of $\mathbf{X}^*$ is equal to the investigated input vector $\mathbf{x}$ while the other rows are equal to the row of the design matrix $\mathbf{X}$. The probability distribution of the associated output vector $\mathbf{y}^* \in \mathbb{R}^{N+1}$, according to the GPR framework is

$$\mathbf{y}^* \sim \mathcal{N}(\mathbf{0}, \mathbf{K}^* + \beta\mathbf{I}_{N+1}). \tag{12}$$

The $N+1 \times N+1$ matrix $\mathbf{K}^*$, is the Gram matrix calculated on the design matrix $\mathbf{X}^*$. In the prediction phase the first N element of the vector $\mathbf{y}_i^*,\ i \in 1, \ldots, N$ are fixed to the values of the training samples y_i. The last element of $\mathbf{y}^*$, which is the value of interest in the regression problem, will have a probability distribution that depends on the value taken by the first N entries of the vector and the kernel function evaluations stored in the Gram matrix $\mathbf{K}^*$. Because of the property of the Gaussian multivariate function such conditional posterior probability is still Gaussian and its parameters are given by

$$\mu^* = \boldsymbol{\kappa}^T(\mathbf{K} + \beta\mathbf{I}_N)^{-1}\mathbf{y} \tag{13}$$

$$\sigma^* = k(\mathbf{x}, \mathbf{x}) - \boldsymbol{\kappa}^T(\mathbf{K} + \beta\mathbf{I}_N)^{-1}\boldsymbol{\kappa} \tag{14}$$

where μ^* and σ^* denote the mean and variance, respectively, and κ is defined as $\boldsymbol{\kappa} \in \mathbb{R}^N \triangleq [k(\mathbf{x}_i, \mathbf{x}), \ldots, k(\mathbf{x}_N, \mathbf{x})]$. By defining $\boldsymbol{\alpha} \in \mathbb{R}^N \triangleq (\mathbf{K} + \beta\mathbf{I}_N)^{-1}\mathbf{y}$ (13) can be expressed in the form of (4) as: $\boldsymbol{\alpha}^T\boldsymbol{\kappa}$. Since in this work, we were interested in a point estimation of the target values, the value of the mean was taken as prediction output for the GPR.

IV. Adiabatic Quantum Kitchen Sinks

An implementation of RKS employing parametric quantum circuits as random feature generators has been recently proposed [28]. In such a procedure, data are encoded in the parameters of quantum circuit, i.e., the angle rotations of the quantum gates that make up the circuit, and the randomization in the feature generation process is obtained by carrying out the measurement on the quantum state after the application of the quantum circuit. They key aspect of this method is that the data encoding is done by a linear function, therefore the nonlinear modeling achieved in the feature transformation is attributable to quantum computation effects. In the QA-based AQKS, implementation data is encoded in a QUBO problem that is then solved with QA. The resulting solution after the Hamiltonian evolution is then used to construct the transformed feature vectors. The encoding is determined by E random matrices $\mathbf{A}_i,\ i = 1, \ldots, E$ of size $q \times d$ and E random vectors $\mathbf{b}_i,\ i = 1, \ldots, E$ of size q, where q is a hyperparameter that controls the dimension of the resulting QUBO problem and d is the dimension of the input feature space. For each training sample $\mathbf{x}_i$, E random vectors $\mathbf{h}_i^e$ are generated with the formula

$$\mathbf{h}_i^e = \mathbf{A}_e\mathbf{x}_i + \mathbf{b}_e \tag{15}$$

where the subscripts i and the superscripts e are used to denote the random vector $\mathbf{h}$ generated from trainig sample i at episode e. Each vector $\mathbf{h}_i^e$ is then encoded in a QUBO problem of size q with the following rule:

$$Q_l = \mathbf{h}_{i,l}^e \tag{16}$$

$$Q_{l,m} = \mathbf{h}_{i,l}^e\mathbf{h}_{i,m}^e \tag{17}$$

with $l, m \in \{1, \ldots, q\}$. At the end of the annealing evolution the vector $\phi(\mathbf{x}_i, \mathbf{A}_e, \mathbf{b}_e)$ of length q is obtained by performing a measurement process and by normalizing by a factor $1/E$. The transformed feature vector $\mathbf{z}_i$ of size $E \times q$ is then obtained by concatenating the E vectors $\{\phi(\mathbf{x}_i, \mathbf{A}_e, \mathbf{b}_e)\ e = 1, \ldots, E\}$. The

Algorithm 1: AQKS Feature Vectors Generation.

Input parameters: training samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, p($\mathbf{A}$) and p($\mathbf{b}$) ,E,q
Output: transformed feature vectors $\mathbf{z}_1, \ldots, \mathbf{z}_N$
sample $\mathbf{A}_1, \ldots, \mathbf{A}_E$ and $\mathbf{b}_1, \ldots, \mathbf{b}_E$ from p($\mathbf{A}$) and p($\mathbf{b}$)
for $i = 1, \ldots, N$ **do**
  for $e = 1, \ldots, E$ **do**
    apply encoding $\mathbf{h}_i^e = \mathbf{A}_e \mathbf{x}_i = \mathbf{b}_e$ encode $\mathbf{h}_i^e$ in a QUBO weight matrix obtain $\phi(\mathbf{x}_i, \mathbf{A}_e, \mathbf{b}_e)$ by performing measurement and normalization after the annealing evolution
  end
  Apply concatenation to the vectors $\phi(\mathbf{x}_i, \mathbf{A}_e, \mathbf{b}_e)$ to get $\mathbf{z}_i = [\phi(\mathbf{x}_1, \mathbf{A}_1, \mathbf{b}_1), \ldots, \phi(\mathbf{x}_i, \mathbf{A}_E, \mathbf{b}_E)]$
end

encoding procedure is again linear and therefore any nonlinearity in the data transformation comes from the QA process. The complete algorithmic workflow for generating the transformed AQKS, defined by Noori et al. [28], is outlined in Algorithm 1 for convenience

The distribution p($\mathbf{A}$) is generally a multivariate Gaussian where each element of A follows a normal distribution $\mathcal{N}(\mu_a, \sigma_a)$ while p($\mathbf{b}$) is a uniform distribution. In our experiments, for each annealing cycle a total of 1000 readouts were considered by setting the parameter *num_reads* in the sampling function from the D-Wave software accordingly. The final value was obtained by doing a weighted average over the obtained samples using as weighting factor the relative occurrence of each vector.

The workflow of AQKS requires the solving of $N \times E$ QUBO problems of size q to generate the transformed feature vectors. The values of the parameters used in the experiments in this work were E=50 and q=4 for the NOMAD dataset and E=100 and q=2 for the SeaBAM dataset, whereas for both cases μ_a=0, σ_a=0.01. The vector $\mathbf{b}$ was ignored in the encoding phase. Since the value of q is generally small, then the annealer will be used to solve many problems of small size in which the vast majority of the available physical qubits will remain unused. In this work, therefore we integrate AQKS with parallel QA to run multiple problem instances together to reduce the computational time. The implementation of AQKS with parallel QA will be referred to as parallel AQKS.

V. Parallel QA

When solving a QUBO problem with a D-Wave quantum annealer the problem graph must be *minor-embedded* [32] in the quantum processing unit. This is done because the hardware topology, which is a Chimera topology for the D-Wave 2000Q and a Pegasus topology for Advantage, does not provide a full connectivity on the hardware graph and therefore, it is often necessary to represent a logical qubit with multiple physical qubits. During this process each logical qubit, which corresponds to a binary variable in the QUBO model, is mapped to a group of connected qubits, which are referred to as a *chain*. The first step in the minor embedding process is the construction of the problem graph $G(V, E)$, in which each of the nodes in V represent a binary variable in the QUBO problem and for each quadratic term in the QUBO a weighted edge with weight equal to the corresponding quadratic coefficient is added. The problem graph is then minor-embedded in the graph defined by the hardware topology. After that, a subgraph of the quantum hardware topology will be then assigned to the problem and the solver will start the annealing procedure on the qubits of such subgraph. In some cases, especially if the problem is of small dimension, it will happen that many of the available qubits will remain unused during the annealing process. Starting from this observation, parallel QA [29] was proposed in order to make better use of the available quantum hardware, considering that two or more independent QUBO problem can be solved together in the same annealing cycle. Let us in fact consider two QUBO problems Q^1 and Q^2, of size m and n, respectively. For the sake of convenience in the notation, let us also denote the variables of Q^1 as $\{a_1, \ldots, a_m\} \in \{0,1\}^m$, and those of Q^2 as $\{a_{m+1}, \ldots, a_{m+n}\} \in \{0,1\}^n$. Now let us consider the QUBO problem $Q^* \triangleq Q^1 + Q^2$, whose variables will then be $a_1, \ldots, a_{m+n} \in \{0,1\}^{m+n}$. It is easy to verify from the problem definition that the minimum of Q^* is equal to the sum of the minimum of Q^1 and Q^2. Moreover, the optimal solution of Q^* will preserve the optimal solutions of Q^1 and Q^2, i.e., the first m variables of the optimal solution of Q^* will be equal to the optimal solution of Q^1 whereas the remaining n variables will be equal to the optimal solution of Q^2. The problem graph related to Q^*, since there are no edges between a_i and a_j with $i \in \{1, \ldots, m\}$ and $j \in \{m+1, \ldots, m+n\}$, will be composed by two independent graphs that are identical to the problem graphs of Q^1 and Q^2. This reasoning could be extended to more than two problems, thus setting the theoretical basis for solving multiple QUBO problems together.

The structure of the encoding problem defined in Section IV is a fully connected graph of size q. Each of the $N \times E$ problems that are needed to generate the feature vectors has the same graph structure, therefore the same embedding scheme can be used when solving together the same number of problems. By solving multiple QUBO problems in parallel we therefore managed to obtain the feature transformation for 20 samples in each annealing cycle. The complete workflow for the proposed parallel implementation of AQKS is outlined in Algorithm 2.

In the pseudocode of Algorithm 2, it was assumed that the number of training sample N was a multiple of the number of samples processed in each annealing cycle, *samples_per_run*. If this is not the case, i.e., $N = p * samples\_per\_run + r$, with $p, r \in \mathbb{N}$ and $0 < r < samples\_per\_run$, the algorithm will run with $num\_iteration = p + 1$: The first p iterations will follow the procedure described by Algorithm 2, while the last one will iterate the for loop over the variable n over $1, \ldots, r$ instead of $1, \ldots, samples\_per\_run$.

VI. Experimental Validation

A. Datasets

The experimental validation in this work has been carried out on two real RS dataset related to water chlorophyll concentration [33].

Algorithm 2: Proposed Implementation Parallel AQKS Implementation.

Input: training samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, probability distributions p($\mathbf{A}$), p($\mathbf{b}$), E, q, number of samples to run in each annealing cycle *samples_per_run*
sample $\mathbf{A}_1, \ldots, \mathbf{A}_E$ and $\mathbf{b}_1, \ldots, \mathbf{b}_E$ from $p(\mathbf{A})$ and $p(\mathbf{b})$, respectively.
define *num_iterations* $\triangleq N/samples\_per\_run$
for $i = 1, \ldots, num\_iterations$ **do**
  initialize Q_i as an empty QUBO problem
  for $n = 1, \ldots, samples\_per\_run$ **do**
    define $c \triangleq samples\_per\_run * (i-1) + n$
    for $e = 1, \ldots, E$ **do**
      calculate random vector $h_c^e = \mathbf{A}_e \mathbf{x}_c + \mathbf{b}_e$
      add to the QUBO Q_i the variables $a_{E*q*(n-1)+(e-1)*q+1}, \ldots, a_{E*q*(n-1)+(e-1)*q+q}$
      set the linear coefficients for $\{a_{E*q*(n-1)+(e-1)*q+1}, \ldots, a_{E*q*(n-1)+(e-1)*q+q}\}$ as $\mathbf{h}^e_{c,1}, \ldots, \mathbf{h}^e_{c,q}$, respectively
      set the the quadratic coefficients $Q_{E*q*(n-1)+(e-1)*q+j, E*q*(n-1)+(e-1)*q+k}$ as $\mathbf{h}^e_{c,j}\mathbf{h}^e_{c,k}$ for $k, j \in \{1, \ldots, q\}$
    end
  end
  Run the annealing evolution on the problem Q_i
  Perform the measurement process and average among the different sample readouts provided by the annealer
  Normalize the obtained vector $\mathbf{z}$ of size $samples\_per\_run * E * q$ vector by a factor 1/E
  Obtain the transformed feature vectors for samples $(i-1)*samples\_per\_run+1, \ldots, (i-1)*samples\_per\_run+samples\_per\_run$
  by considering the elements of $\mathbf{z}$ $\{E*(l-1)*q+1, \ldots, E*l*q\}$ with $l \in \{1, \ldots, samples\_per\_run\}$
end

1) *SEABAM* [34] (SeaWiFS Bio-optical Algorithm Mini-Workshop) The first dataset used contains 919 in situ measurements of chlotophyll concentration in water taken from several locations in U.S. and Europe. However, due to some missing data value only 793 samples were used in the experiments. The measurements were carried out with the Sea-viewing Wide Field-of- view Sensor (SeaWiFS) at five different wavelengths (412, 443, 490, 510, and 555 nm) and the chlorophyll concentration takes values in the range 0.019 and 32.787 mg/m^3.
2) *NOMAD* [35] (NASA bio-Optical Marine Algorithm Dataset) The second dataset used is also an in situ dataset and contains several bioptical data information such as surface irradiances, water-leaving radiances, diffuse downwelling attenuation coefficients, and chlorophyll concentration values. In this work data taken at five different wavelengths (411, 443, 489, 510, and 555 nm) were used as input features vectors for the regression algorithms. Specifically, for each spectral band the corresponding feature value was taken as the ratio between the corresponding spectral water-leaving radiance and the spectral surface irradiance [2]. For the experimental part of this work, a total of 1210 measurements were used and the chlorophyll concentration value ranged between 0.017 and 70.21 mg/m^3.

For the training phase in both datasets, the values of both the feature vector and the target value were converted to the logarithmic domain. The reason for this is that the values of the bio-physical quantities were assumed to be log-normally distributed [36].

B. Implementation Details

For each dataset the two regression methods (SVR, GPR) implemented with the parallel AQKS kernel approximation were tested on ten different randomly sampled training and test sets of size 200 each. On each of these run a classical implementation of the regression algorithm using a RBF kernel and a RKS kernel approximation were tested and their results in terms of R2 score and mean squared error (MSE) were compared as a benchmark. The results achieved in terms of R2 score and MSE by the three different kernel implementation were then compared.

In each run the hyperparameters of the regression algorithms were tuned by running a exhaustive grid search defined over a discrete hyperparameter space on a five-fold validation on the training set. Specifically, the training set has been divided in five different subsets (folds) and each hyperparameter configuration was tested on each fold after being trained on the remaining four other. The configuration that achieved the highest average R2 score over the five different folds was selected. Since the parameters of parallel AQKS kernel approximation were not optimized empirically because of the computational burden, it was not performed an optimization of the kernel parameter γ for the classical and RKS-based algorithm. Such value was set to 1 for the SVR and 2 for the GPR in the classical case, whereas it was set to 1 for both SVR and GPR in the RKS implementation. The number of components in the RKS algorithm was set to 50. All the classical algorithms have been implemented using the python library scikit-learn [37]. The hyperparameter spaces for the learning algorithms are as follows.

1) *SVR:* $C : [2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}2^{-2}, 2^{-1}, 1, 2, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8]$, $\epsilon : [10^{-3}, 10^{-2}, 10^{-1}]$
2) *GPR:* Noise parameter β: $[10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$

As indicated in Section VI-A, the training phase has been conducted by considering the logarithm values of both the input vector and the target value. The trained prediction function then provided a target value estimation in the logarithmic domain. For the evaluation of the chosen performance metrics two different setting were considered: In the first one, the comparison between the predicted and the actual values was carried out by comparing the value provided by the prediction function and the logarithm of the target value, whereas in the second setting the evaluation was conducted by considering the original target value and the prediction value in the original domain (obtained by exponentiation). In the following these two settings

TABLE I
RESULTS ACHIEVED BY THE DIFFERENT KERNEL IMPLEMENTATIONS IN THE LOGARITHM SETTING FOR THE NOMAD DATASET

	R2						MSE					
	AQKS		RKS		Classical		AQKS		RKS		Classical	
Experimental run	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR
1	0.8547	0.8464	0.8661	0.8326	0.8889	0.9171	0.4041	0.4271	0.3724	0.4655	0.3089	0.2306
2	0.8467	0.8653	0.8556	0.9031	0.8666	0.8966	0.4747	0.4171	0.447	0.3001	0.4131	0.3202
3	0.8522	0.8305	0.865	0.8488	0.8512	0.8774	0.3625	0.4159	0.3312	0.371	0.3651	0.3009
4	0.8351	0.8157	0.8628	0.845	0.8373	0.8282	0.4019	0.4489	0.3343	0.3776	0.3964	0.4185
5	0.8183	0.8362	0.8278	0.834	0.86	0.8782	0.4801	0.4328	0.455	0.4387	0.3699	0.3218
6	**0.841**	0.8319	0.8296	0.8181	0.8331	0.8703	**0.3978**	0.4208	0.4263	0.4552	0.4176	0.3245
7	**0.8752**	0.8642	0.8594	0.8703	0.8503	0.8785	**0.3892**	0.4235	0.4385	0.4045	0.4671	0.3789
8	0.8496	0.8676	0.8668	0.8863	0.8852	0.8958	0.3942	0.347	0.3492	0.298	0.301	0.2731
9	0.8214	0.8388	0.8421	0.8718	0.844	0.8729	0.4865	0.439	0.4302	0.3493	0.4251	0.3462
10	0.8519	0.8605	0.8646	0.8316	0.8709	0.8904	0.4182	0.394	0.3824	0.4755	0.3646	0.3095
Average	0.84461	0.84571	0.85398	0.85416	0.85875	0.88054	0.42092	0.41661	0.39665	0.39354	0.38288	0.32242
Standard deviation	0.0158	0.017	0.0144	0.0261	0.0181	0.022	0.0412	0.0271	0.0457	0.062	0.0492	0.0495

The experimental runs in which the AQKS implementation achieved better results than the classical implementations are highlighted with a bold font.

TABLE II
RESULTS ACHIEVED BY THE DIFFERENT KERNEL IMPLEMENTATIONS IN THE ORIGINAL SETTING FOR THE NOMAD DATASET

	R2						MSE					
	AQKS		RKS		Classical		AQKS		RKS		Classical	
Experimental run	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR
1	**0.4694**	0.573	0.3067	0.3723	0.3445	0.7877	**29.7181**	23.917	38.8359	35.1613	36.7163	11.8921
2	**0.3639**	0.4158	0.2223	0.4802	0.3065	0.4924	**43.6422**	40.0798	53.3583	35.6625	47.5796	34.8237
3	0.5114	**0.6572**	0.5939	0.6234	0.4595	0.6185	12.4689	**8.74798**	10.3644	9.6101	13.7923	9.7345
4	**0.4678**	0.484	0.4515	0.5191	0.3411	0.1149	**6.18735**	5.99895	6.3763	5.5906	7.6599	10.2898
5	**0.5856**	**0.6621**	0.3673	0.2915	0.3698	0.5653	**21.9323**	**17.8812**	33.4848	37.4971	33.3539	23.0084
6	**0.5017**	0.5769	0.413	0.4977	0.3062	0.662	**15.7916**	13.4079	18.6024	15.9181	21.9866	10.7112
7	**0.6584**	**0.7147**	0.4417	0.5533	0.3103	0.6537	**20.0019**	**16.7071**	32.6923	26.16	40.3864	20.2803
8	**0.65**	0.6482	0.5827	0.731	0.6066	-0.0895	**14.9709**	15.0517	17.8522	11.5096	16.8273	46.6078
9	0.2984	**0.4916**	0.3528	0.3116	0.1953	0.3427	36.9161	**26.7502**	34.0534	36.2232	42.3389	34.5869
10	0.5722	0.6509	0.5917	0.5946	0.6025	0.7002	20.7195	16.9072	19.7742	19.6351	19.2548	14.5187
Average	**0.50788**	**0.58744**	0.43236	0.49747	0.38423	0.48479	**22.2349**	**18.5449**	26.5394	23.2968	27.9896	21.6453
Standard deviation	0.1099	0.0917	0.1206	0.1328	0.1264	0.2663	10.8828	9.265	13.6581	11.7399	13.0387	12.2649

The experimental runs in which the AQKS implementation achieved better results than the classical implementations are highlighted with a bold font.

TABLE III
RESULTS ACHIEVED BY THE DIFFERENT KERNEL IMPLEMENTATIONS IN THE LOGARITHM SETTING FOR THE SEABAM DATASET

	R2						MSE					
	AQKS		RKS		Classical		AQKS		RKS		Classical	
Experimental run	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR
1	0.8984	0.8887	0.9091	0.9034	0.912	0.9131	0.1987	0.2178	0.1777	0.189	0.1721	0.17
2	0.9039	0.8942	0.91	0.9107	0.9172	0.9249	0.2188	0.2408	0.2049	0.2033	0.1886	0.171
3	0.888	0.8953	0.8849	0.8877	0.8901	0.8967	0.2575	0.2409	0.2648	0.2582	0.2527	0.2376
4	**0.8975**	**0.8922**	0.8732	0.8346	0.8462	0.8594	**0.2184**	**0.2296**	0.2701	0.3522	0.3275	0.2994
5	**0.8788**	0.8818	0.8429	0.8163	0.8782	0.8989	**0.2488**	0.2426	0.3225	0.3771	0.2501	0.2076
6	0.888	0.8857	0.8826	0.8986	0.9204	0.9227	0.2289	0.2336	0.2399	0.2072	0.1627	0.158
7	0.8744	0.8677	0.8858	0.888	0.8897	0.8861	0.2367	0.2493	0.2151	0.211	0.2079	0.2146
8	0.8903	0.8775	0.8982	0.898	0.9065	0.9082	0.2062	0.2302	0.1914	0.1917	0.1758	0.1726
9	0.8551	0.8918	0.9019	0.8244	0.8974	0.8958	0.3369	0.2516	0.2281	0.4083	0.2385	0.2423
10	**0.8829**	0.8676	0.877	0.8653	0.8403	0.8787	0.2453	0.2773	0.2578	0.2823	0.3347	0.2542
Average	0.88573	0.88425	0.88656	0.8727	0.8898	0.89845	0.23962	0.24137	0.23723	0.26803	0.23106	0.21273
Standard deviation	0.0134	0.0098	0.019	0.0334	0.0265	0.0192	0.037	0.0153	0.041	0.0788	0.0587	0.0435

The experimental runs in which the AQKS implementation achieved better results than the classical implementations are highlighted with a bold font.

will be referred to as *logarithm* setting and *original* setting, respectively.

VII. RESULTS

The results on the NOMAD dataset in the logarithm and original setting are reported in Tables I and II, respectively. Tables III and IV show the results for the SEABAM dataset (logarithm and original setting, respectively). In the logarithm domain the three kernel implementations performed similarly in terms of R2 score and MSE on both datasets with the classical GPR implementation obtaining slightly better results overall. Interesting insights can be considered by analyzing the results in the original domain: For the NOMAD dataset the parallel AQKS implementation achieved the best average results on both R2 score and MSE. In the SEABAM dataset, the situation was more diverse: The classical SVR implementation achieved the best R2 score, whereas the classical GPR obtained the worst performances on the same evaluation metric. The parallel AQKS GPR performed slightly better than RKS implementation while for the SVR the latter kernel approximation method performed slightly better. Regarding the MSE, the results were also similar with the classical SVR and GPR obtaining the best and worst results, respectively. It is also worth noting that the proposed parallel AQKS implementation never obtained a negative value for the R2 score across the various experimental runs, while

TABLE IV
RESULTS ACHIEVED BY THE DIFFERENT KERNEL IMPLEMENTATIONS IN THE ORIGINAL SETTING FOR THE SEABAM DATASET

	R2						MSE					
	AQKS		RKS		Classical		AQKS		RKS		Classical	
Experimental run	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR	SVR	GPR
1	0.6114	0.5644	0.8979	0.6583	0.7952	0.7682	2.54541	2.85357	0.6687	2.2384	1.34169	1.51868
2	0.412	0.4377	0.6727	0.8462	0.7979	0.8619	6.5645	6.2784	3.6544	1.7172	2.25637	1.54226
3	**0.4354**	**0.5463**	0.3898	0.4376	0.4121	0.4186	**6.29623**	**5.05951**	6.8051	6.2719	6.55649	6.4839
4	**0.4112**	**0.3947**	0.2751	0.171	0.1877	0.1918	**6.69937**	**6.88736**	8.2478	9.4322	9.2416	9.19557
5	0.522	0.5035	0.4217	0.5436	0.6415	0.7067	3.09488	3.21482	3.7441	2.9551	2.32111	1.89872
6	0.3853	0.4166	0.3496	0.4317	0.5207	0.6046	8.14113	7.72548	8.6132	7.5255	6.34713	5.23573
7	0.2644	0.288	0.343	0.5419	0.3899	-2.1175	8.40368	8.13365	7.505	5.2329	6.96937	35.6133
8	0.5801	0.6298	0.7599	0.6463	0.7777	0.6915	1.90872	1.68274	1.0914	1.608	1.01051	1.40248
9	**0.5479**	0.542	0.5044	-0.3097	0.5025	0.6241	**3.95809**	4.01001	4.3391	11.4678	4.35617	3.29168
10	0.6027	0.6699	0.6949	0.6659	0.709	0.4923	1.67502	**1.3919**	1.2862	1.4084	1.22674	2.14073
Average	0.47724	**0.49929**	0.5309	0.46328	0.57342	0.32422	4.9287	**4.72374**	4.5955	4.98574	4.16272	6.83231
Standard deviation	0.1074	0.1098	0.2001	0.3093	0.1951	0.8338	2.4448	2.3341	2.8837	3.4134	2.7884	9.9055

The experimental runs in which the AQKS implementation achieved better results than the classical implementations are highlighted with a bold font.

the RKS-based implementation obtained a negative score once with GPR algorithm (experimental run 9 on the SEABAM in the original setting) and the classical GPR twice (experimental run 7 for the SEABAM and experimental run 8 for the NOMAD, both in the original setting). Another interesting fact can be observed by analyzing the best R2 score achieved across the various experimental runs. In the original setting for the SEABAM dataset both the RKS-based and classical algorithm always obtained a higher best R2 across the different runs with respect to the AQKS even when the AQKS achieved a higher average score. This fact might indicate a better robustness of the AQKS in terms of generalization with respect to new dataset sampling, however further research is needed to verify this hypothesis.

VIII. CONCLUSION

The objective of this work was to develop a AQKS kernel approximation implementation on a quantum annealer using parallel QA for regression applications. The choice of using a parallel implementation was motivated by the high number of QUBO problems that are needed in the workflow. The proposed implementation managed to achieve results comparable to those obtained by classical kernel methods and the traditional RKS kernel approximation algorithm, which could be indicative of its potential. The maximum number of samples obtained on each annealing cycle, given the number of epochs E and the number of qubits q, is limited by the size of the quantum hardware. In our work we managed to obtain 20 transformed feature vectors in each annealing cycle, which makes the process unfeasible for large datasets. The problem graph for the parallel annealing, since is composed of many independent smaller subgraphs, is sparsely connected and therefore, might scale well with a increased availability of physical qubits in future GA hardware. Further research could also be conducted to improve upon the proposed implementation. For instance, the samples that are selected on each annealing cycle were chosen in a sequential approach based on their sample index in the dataset; further research could investigate a way to select the samples to be considered in the same annealing cycle to increase the performances. The code associated with this work can be found at this GitHub repository.[1]

[1]GitHub repository: https://gitlab.jsc.fz-juelich.de/sdlrs/quantum-kernel-estimation-parallel-random-kitchen-sinks

REFERENCES

[1] Y. Bazi and F. Melgani, "Semisupervised PSO-SVM regression for biophysical parameter estimation," *IEEE Trans. Geosci. Remote Sens.*, vol. 45, no. 6, pp. 1887–1895, Jun. 2007.

[2] Y. Bazi, N. Alajlan, and F. Melgani, "Improved estimation of water chlorophyll concentration with semisupervised Gaussian process regression," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 7, pp. 2733–2743, Jul. 2012.

[3] H. Zhan, P. Shi, and C. Chen, "Retrieval of oceanic chlorophyll concentration using support vector machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 12, pp. 2947–2951, Dec. 2003.

[4] F. DelFrate, A. Ortenzi, S. Casadio, and C. Zehner, "Application of neural algorithms for a real time estimation of ozone profiles from gome measurements," in *Proc. Scanning Present Resolving Future, IEEE Int. Geosci. Remote Sens. Symp.*, 2001, vol. 6, pp. 2668–2670.

[5] H. Aghighi, M. Azadbakht, D. Ashourloo, H. S. Shahrabi, and S. Radiom, "Machine learning regression techniques for the silage maize yield prediction using time-series images of landsat 8 oli," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 12, pp. 4563–4577, Dec. 2018.

[6] X. Wang, L. Ma, and X. Wang, "Apply semi-supervised support vector regression for remote sensing water quality retrieving," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2010, pp. 2757–2760.

[7] A. Rabe, S. van der Linden, and P. Hostert, "Simplifying support vector machines for regression analysis of hyperspectral imagery," in *Proc. 1st Workshop Hyperspectral Image Signal Process.: Evol. Remote Sens.*, 2009, pp. 1–4.

[8] G. Mateo-García, V. Laparra, and L. Gómez-Chova, "Optimizing kernel ridge regression for remote sensing problems," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2018, pp. 4007–4010.

[9] Y. Bazi, N. Alajlan, F. Melgani, H. AlHichri, and R. R. Yager, "Robust estimation of water chlorophyll concentrations with Gaussian process regression and iowa aggregation operators," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 7, pp. 3019–3028, Jul. 2014.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[11] K. P. Murphy, *Machine Learning : A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2013. [Online]. Available: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2

[12] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. 20th Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 1177–1184.

[13] A. Rahimi and B. Recht, "Uniform approximation of functions with random bases," in *Proc. 46th Annu. Allerton Conf. Commun., Control, Comput.*, 2008, pp. 555–561.

[14] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge, U.K.: Cambridge Univ. Press, Dec. 2010.

[15] A. Montanaro, "Quantum algorithms: An overview," *NPJ Quantum Inf.*, vol. 2, no. 1, Jan. 2016, Art. no. 15023, doi: 10.1038/npjqi.2015.23.

[16] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

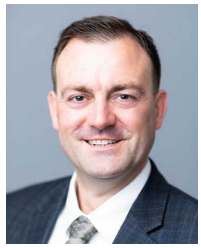
[17] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, New York, NY, USA, 1996, pp. 212–219, doi: 10.1145/237814.237866.

[18] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, Aug. 2018, Art. no. 79, doi: 10.22331/q-2018-08-06-79.

[19] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
[20] N. Mishra et al., "Quantum machine learning: A review and current status," in *Proc. Data Manage., Anal. Innov.*, 2021, pp. 101–145.
[21] M. Schuld and F. Petruccione, "Quantum machine learning," in *Encyclopedia of Machine Learning and Data Mining*. Berlin, Germany: Springer, 2017, pp. 1034–1043, doi: 10.1007/978-1-4899-7687-1_913.
[22] V. Dunjko and H. J. Briegel, "Machine learning and artificial intelligence in the quantum domain: A review of recent progress," *Rep. Prog. Phys.*, vol. 81, no. 7, Jun. 2018, Art. no. 074001, doi: 10.1088/1361-6633/aab406.
[23] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Phys. Rev. Lett.*, vol. 122, no. 4, Feb. 2019, Art. no. 040504.
[24] A. Sebastianelli, D. A. Zaidenberg, D. Spiller, B. L. Saux, and S. L. Ullo, "On circuit-based hybrid quantum neural networks for remote sensing imagery classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 565–580, 2022.
[25] S. Otgonbaatar and M. Datcu, "Classification of remote sensing images with parameterized quantum gates," *IEEE Geosci. Remote Sens. Lett.*, vol. 19, 2022, Art. no. 8020105.
[26] P. Gawron and S. Lewiński, "Multi-spectral image classification with quantum neural network," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2020, pp. 3513–3516.
[27] C. M. Wilson et al., "Quantum kitchen sinks: An algorithm for machine learning on near-term quantum computers," 2018. [Online]. Available: https://arxiv.org/abs/1806.08321
[28] M. Noori et al., "Analog-quantum feature mapping for machine-learning applications," *Phys. Rev. Appl.*, vol. 14, no. 3, Sep. 2020, Art. no. 034034, doi: 10.1103/physrevapplied.14.034034.
[29] E. Pelofske, G. Hahn, and H. N. Djidjev, "Parallel quantum annealing," *Sci. Rep.*, vol. 12, no. 1, Mar. 2022, Art. no. 4499, doi: 10.1038/2Fs41598-022-08394-8.
[30] F. Barahona, "On the computational complexity of ising spin glass models," *J. Phys. A: Math. Gen.*, vol. 15, no. 10, pp. 3241–3253, Oct. 1982, doi: 10.1088/0305-4470/15/10/028.
[31] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA, Springer2000, doi: 10.1007/978-1-4757-3264-1.
[32] V. Choi, "Minor-embedding in adiabatic quantum computation: I. the parameter setting problem," *Quantum Inf. Process.*, vol. 7, pp. 193–209, 2008. [Online]. Available: https://arxiv.org/abs/0804.4884v1
[33] D. Pastorello, *Concise Guide to Quantum Machine Learning*. Singapore: Springer, 2023, doi: 10.1007/978-981-19-6897-6.
[34] J. E. O'Reilly et al., "Ocean color chlorophyll algorithms for seawifs," *J. Geophys. Res.: Oceans*, vol. 103, no. C11, pp. 24937–24953, 1998, doi: 10.1029/98JC02160.
[35] P. J. Werdell and S. W. Bailey, "An improved in-situ bio-optical data set for ocean color algorithm development and satellite data product validation," *Remote Sens. Environ.*, vol. 98, pp. 122–140, 2005.
[36] J. W. Campbell, "The lognormal distribution as a model for the bio-optical versatility of the sea," *J. Geophys. Res.*, vol. 100, pp. 13237–13254, 1995.
[37] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

Edoardo Pasetto received the B.Sc. and M.Sc. degrees in information and communication engineering from the University of Trento, Trento, Italy, in 2019 and 2021, respectively. He is currently working toward the Ph.D. degree in physics with Forschungszentrum Jülich, Germany, and RWTH Aachen University, Aachen, Germany.

His main research interest include the application of hybrid quantum-classical machine learning frameworks to RS applications.

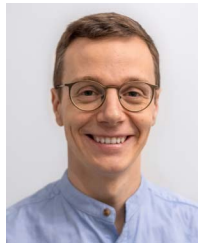


Morris Riedel (Member, IEEE) received the Ph.D. degree in parallel and distributed systems from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, in 2012.

He is currently a Full Professor of high-performance computing with an emphasis on parallel and scalable machine learning with the School of Natural Sciences and Engineering, the University of Iceland, Reykjavik, Iceland. He has worked in data-intensive parallel and distributed systems since 2004, and since then, he has held various positions at the Juelich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany. In addition, he is the Head of the joint High Productivity Data Processing research group between the Juelich Supercomputing Centre and the University of Iceland. Since 2020, he is also the EuroHPC Joint Undertaking governing board member for Iceland. His online YouTube and university lectures include High-Performance Computing – Advanced Scientific Computing, Cloud Computing and Big Data – Parallel and Scalable Machine and Deep Learning, as well as Statistical Data Mining. In addition, he has performed numerous hands-on training events in parallel and scalable machine and deep learning techniques on cutting-edge HPC systems. He has authored or coauthored extensively in the areas of his interest. His research interests include high-performance computing, remote sensing applications, medicine and health applications, pattern recognition, image processing, and data sciences.

Kristel Michielsen received the Ph.D. degree in physics for her work on the simulation of strongly correlated electron systems from the University of Groningen, Groningen, the Netherlands, in 1993.

Since 2009, she is Group Leader of the research group Quantum Information Processing, Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich, Jülich, Germany, and Professor of quantum information processing with RWTH Aachen University, Aachen, Germany. She and her group have ample experience in performing large-scale simulations of quantum systems. With her group and a team of international collaborators, she set the world record in simulating a quantum computer with 48 qubits. In 2019, she participated in a research collaboration that proved Google's quantum supremacy. She is leading and building up the Jülich UNified Infrastructure for Quantum computing (JUNIQ), JSC. Her research interests include classical simulations of electrodynamics, quantum mechanics, quantum computing, quantum computing architectures, quantum algorithms, quantum benchmarking, and modular quantum-HPC hybrid computing.



Gabriele Cavallaro (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in telecommunications engineering from the University of Trento, Trento, Italy, in 2011 and 2013, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Iceland, Reykjavik, Iceland, in 2016.

From 2016 to 2021, he has been the Deputy Head of the "High Productivity Data Processing" (HPDP) research group, Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich, Jülich, Germany. Since 2022, he is the Head of the "AI and ML for Remote Sensing" Simulation and Data Lab, JSC, and an Adjunct Associate Professor with the School of Natural Sciences and Engineering, University of Iceland. From 2020 to 2023, he held the position of Chair for the High-Performance and Disruptive Computing in Remote Sensing (HDCRS) Working Group under the IEEE GRSS Earth Science Informatics Technical Committee (ESI TC). In 2023, he took on the role of Co-chair for the ESI TC. Concurrently, he serves as Visiting Professor with the Φ-lab, European Space Agency (ESA), where he contributes to the Quantum Computing for Earth Observation (QC4EO) initiative. His research interests include remote sensing data processing with parallel machine learning algorithms that scale on distributed computing systems and cutting-edge computing technologies, including quantum computers.

Dr. Cavallaro was the recipient of the IEEE GRSS Third Prize in the Student Paper Competition of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS) 2015 (Milan - Italy). He has been serving as an Associate Editor for IEEE TRANSACTIONS ON IMAGE PROCESSING (TIP) since October 2022.