Kernel Approximation on a Quantum Annealer for Remote Sensing Regression Tasks

Edoardo Pasetto ¹, Morris Riedel ¹, Kristel Michielsen ¹, and Gabriele Cavallaro ²

¹Affiliation not available
²Forschungszentrum Jülich

October 31, 2023

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 (RS) community. This paper 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 (SVR) and Gaussian Process Regression (GPR) 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 from a classical implementation of kernel-based algorithms and a Random Kitchen Sinks (RKS) implementation. On average, the parallel AQKS achieved comparable results to the benchmark methods, indicating its potential for future applications.
Kernel Approximation on a Quantum Annealer for Remote Sensing Regression Tasks

Edoardo Pasetto, Morris Riedel, Member, IEEE, Kristel Michielsen, and Gabriele Cavallaro, 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 (RS) community. This paper 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 (SVR) and Gaussian Process Regression (GPR) algorithms. To evaluate its performance, a regression problem related to estimating chlorophyll concentration is considered. The proposed algorithm was tested on two real-world datasets and its results were compared with those obtained from a classical implementation of kernel-based algorithms and a Random Kitchen Sinks (RKS) implementation. On average, the parallel AQKS achieved comparable results to the benchmark methods, indicating its potential for future applications.

Index Terms—Quantum Computing (QC), Quantum Annealing (QA), regression, Remote Sensing (RS), parallel quantum annealing

I. INTRODUCTION

T he task of estimating biophysical quantities from 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 modelling problem whose objective is to find a relationship between acquired measurements of some specific physical quantities and value of interest [6]. On a formal point of view the objective is to determine a function \( y = f(x) : \mathbb{R}^d \rightarrow \mathbb{R} \), where \( x \in \mathbb{R}^d \) is the input feature vector containing the data of the optical measurements and the scalar \( y \subseteq \mathbb{R} \) 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 \( \{(x_i, y_i), \ i = 1, \ldots, N\} \). Regression tasks in RS have been studied by applying different supervised learning algorithms and among the most popular are SVR [7], [8], Kernel Ridge Regression (KRR) [9] and GPR [10]. A common feature of these methods is the usage of a kernel function \( k(x, x') \), which allows to calculate the dot product between a non-linear projection of the input vectors in a transformed feature space taking as argument the original input vectors, i.e. \( k(x, x') = \phi(x)'^T \phi(x') \), where \( \phi(.) \) is a non-linear 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 [11], [12]. The main characteristic of this procedure is that it is not necessary to know the non-linear 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 [13]. Starting from this observation Rahimi et al. proposed the RKS kernel approximation algorithm, which approximates the kernel function by using randomized features [13], [14]. This procedure, also known as Random Fourier Features, therefore does not employ a kernel function but instead explicitly generates transformed feature vectors through randomization and use them on a linear learning algorithm.

Quantum Computing (QC)[15], [16] is a computational model based upon the properties of quantum mechanics that was theoretically proven to have the potential to outperform any classical computer on some specific tasks [17], [18]. However, the availability of a reliable large-scale quantum computer might still be a distant goal [19]. 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) [20], [21], [22], [23], [24]. In the context of RS, QML have been applied to image classification through the usage of a hybrid quantum-classical neural network whose quantum layer was implemented with a parametrized quantum circuit [25], [26], [27].

A QML-based implementation of the Random Fourier features has been recently proposed with gate-based quantum computing [28] and Quantum Annealing (QA) [29]. In the QA-based implementation, also referred to as 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 2 different
kernel-based regression algorithms: SVR and GPR on 2 real RS datasets related to chlorophyll concentration estimation. The results obtained 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 [29] 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 [30] 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 quantum annealing device, application of such a scheme to regression problem with 2 different algorithms, integration of AQKS with parallel quantum annealing 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 a Quadratic Binary Unconstrained Optimization (QUBO), which corresponds to the optimization of the following energy function:

$$\text{min} \sum_{i=1}^{N} \sum_{j=i+1}^{N} a_{i}Q_{ij}a_{j}$$

(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 \(a \in \{0, 1\}^{N} \triangleq [a_{1}, \ldots, a_{N}]\) it is possible to rewrite equation 1 in matrix product form as:

$$\text{min} \ a^{T}Qa$$

(2)

Alternatively, it is also possible to formulate the problem as a Ising spin model [31], 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 positive semi-definite function \(k(x, x')\) can be used as kernel function [13]. 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. \(k(x, x') = k(||x - x'||)\). The formula of the RBF is:

$$k(x, x') = \exp\left(\frac{||x - x'||}{2\gamma}\right)$$

(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 \(x\):

$$f(x) = \sum_{i=1}^{N} \alpha_{i}k(x_{i}, x) + b$$

(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 non-linear modelling 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 \(X\) the \(N \times d\) design matrix in which each of its row corresponds to a training sample, i.e \(X_{i,:} = x_{i}, i = 1, \ldots, N\) and as \(y \in \mathbb{R}^{N}\) the corresponding target vector. Let us also define as \(K\) the \(N \times N\) symmetric matrix that stores the kernel function evaluation between every pair of training sample \(x_{i}\) and \(x_{j}\), i.e. \(K_{ij} = K(x_{i}, x_{j})\).

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 [32], 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 \(\epsilon > 0\) [11]:

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

(5)

The loss function to be minimized is then:

$$C \sum_{n=1}^{N} \mathcal{L}_{\epsilon}(f(x_{n}) - y_{n}) + \frac{1}{2}||w||^{2}$$

(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 [11].

It can be shown that the training of the SVR amounts to the solving of the following constrained optimization problem [11]:

$$L(\alpha, \hat{\alpha}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} (\alpha_{n} - \hat{\alpha}_{n})(\alpha_{m} - \hat{\alpha}_{m})k(x_{n}, x_{m}) +$$

$$- \epsilon \sum_{n=1}^{N} (\alpha_{n} + \hat{\alpha}_{n}) + \sum_{n=1}^{N} (\alpha_{n} - \hat{\alpha}_{n})y_{n}$$

subject to the constraints:

$$\sum_{n=1}^{N} (\alpha_{n} - \hat{\alpha}_{n}) = 0$$

(8a)
with respect to the variables $\alpha_n$ and $\hat{\alpha}_n$ with $i \in 1, \ldots, N$. Once the values of $\alpha_1, \ldots, \alpha_N$ and $\alpha_1, \ldots, \hat{\alpha}_N$ have been determined a prediction on an input sample $x$ can then be made through the formula:

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

(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:

$$b = t_n - \epsilon - \sum_{m=1}^{N} (\alpha_m - \hat{\alpha}_m) k(x_n, x_m)$$

(10)

It is preferable, however, to average over multiple data points in order to get a more stable estimation [11].

### 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 $\mu^*$ and variance $\sigma^*$. In GPR the relationship between the input vectors stored in $X$ and the target values is modelled as a sum between a gaussian multivariate function $\mathcal{N}(0, K)$ and a independent noise component $\mathcal{N}(0, \beta^{-1}I_N)$. 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 [11] the target values assume the following probability distribution:

$$y \sim \mathcal{N}(0, K + \beta I_N)$$

(11)

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

$$y^* \sim \mathcal{N}(0, K^* + \beta I_{N+1})$$

(12)

The $N+1 \times N+1$ matrix $K^*$, is the Gram matrix calculated on the design matrix $X^*$. In the prediction phase the first $N$ element of the vector $y^*_i, i \in 1, \ldots, N$ are fixed to the values of the training samples $y_i$. The last element of $y^*$, 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 by the kernel function evaluations stored in the Gram Matrix $K^*$. Because of the property of the gaussian multivariate function such conditional posterior probability is still gaussian and its parameters are given by:

$$\mu^* = \kappa^T (K + \beta I_N)^{-1} y$$

$$\sigma^* = k(x, x) - \kappa^T (K + \beta I_N)^{-1} \kappa$$

(14)

with $\kappa \in \mathbb{R}^N \triangleq [k(x_1, x), \ldots, k(x_N, x)]$. By defining $\alpha \in \mathbb{R}^N \triangleq (K + \beta I_N)^{-1} y$ equation 13 can be expressed in the form of 4 as: $\alpha^T \kappa$. 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 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 non-linear modelling 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 quantum annealing. The resulting solution after the Hamiltonian evolution is then used to construct the transformed feature vectors. The encoding is determined by $E$ random matrices $A_i$, $i = 1, \ldots, E$ of size $q \times d$ and $E$ random vectors $b_i$, $i = 1, \ldots, E$ 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 $x_i$, $E$ random vectors $h^*_f$ are generated with the formula:

$$h^*_f = A_e x_i + b_e$$

(15)

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

$$Q_{lt} = h^*_f l$$

$$Q_{lm} = h^*_e l$$

(16)

(17)

with $l, m \in \{1, \ldots, q\}$. At the end of the annealing evolution the vector $\phi(x_i, A_e, b_e)$ of length $q$ is obtained by performing a measurement process and by and by normalizing by a factor $1/E$. The transformed feature vector $z_i$ of size $E \times q$ is then obtained by concatenating the $E$ vectors $\{\phi(x_i, A_e, b_e) \mid e = 1, \ldots, E\}$. The encoding procedure is again linear and therefore any non-linearity in the data transformation comes from the quantum annealing process. The complete algorithmic workflow for generating the transformed AQKS, defined by Noori et al. in [29], is outlined in algorithm 1 for convenience:

The distribution $p(A)$ is generally a multivariate gaussian where each element of $A$ follows a normal distribution $\mathcal{N}(\mu, \sigma)$ while $p(b)$ 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
Algorithm 1 AQKS feature vectors generation

Input parameters: training samples \( \{x_1, \ldots, x_N\} \), \( p(A) \) and \( p(b) \).

Output: transformed feature vectors \( z_1, \ldots, z_N \) sampled \( A_1, \ldots, A_E \) and \( b_1, \ldots, b_E \) from \( p(A) \) and \( p(b) \).

For \( i = 1, \ldots, N \) do

For \( e = 1, \ldots, E \) do

Apply encoding \( h_i^e = A_e x_i = b_e \) encode \( h_i^e \) in a QUBO weight matrix obtain \( \phi(x_i, A_e, b_e) \) by performing measurement and normalization after the annealing evolution.

End.

Apply concatenation to the vectors \( \phi(x_i, A_e, b_e) \) to get \( z_i = [\phi(x_1, A_1, b_1), \ldots, \phi(x_i, A_E, b_E)] \).

End.

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 \( E=50, q=4, \mu_n=0, \sigma_n=0.01 \), while the vector \( b \) was ignored in the encoding phase. Since the value of \( q \) is generally small, 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 quantum annealing to run multiple problem instances together to reduce the computational time. The implementation of AQKS with parallel quantum annealing will be referred to as parallel AQKS.

V. PARALLEL QUANTUM ANNEALING

When solving a QUBO problem with a D-Wave quantum annealer the problem graph must be minor-embedded [33] in the Quantum Processing Unit (QPU). This is done because the hardware topology, which is a Chimera topology for the D-Wave 2000Q and a Pegasus topology for Advantage, doesn’t 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 connect 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 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 quantum annealing [30] 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 \( Q^* = Q^1 + Q^2 \) 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 when solving together the same number of problems the same embedding scheme can be used. In this work we solved in parallel 1000 problems of size 4, running each time a problem whose graph was composed of 1000 independent clique graphs of size 4. Since each feature vector generation required the solving of 50 QUBO problems it was possible to get 20 transformed feature vectors for 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, \( \text{samples per run} \).

If this is not the case, i.e \( N = p \times \text{samples per run} + r \), with \( p, r \in \mathbb{N} \) and \( 0 < r < \text{samples per run} \), the algorithm will run with \( \text{num iteration} = p + 1 \): 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 \( 1, \ldots, \text{samples per run} \).

VI. EXPERIMENTAL VALIDATION

A. Datasets

The experimental validation in this work has been carried out on 2 real RS dataset related tho water chlorophyll concentration.

- SEABAM [34] (SeaWiFS Bio-optical Algorithm Mini-Workshop) The first dataset used contains 919 in-situ measurements of chlorophyll concentration in water taken from several locations in U.S. and Europe. The measurements were carried out wit the Sea-viewing Wide Field-of-view Sensor (SeaWiFS) at 5 different wavelengths (412, 443, 490, 510 and 555 nm) and the chlorophyll concentration takes values in the range 0.019 and 32.787 mg/m³.
The results achieved in terms of R2 score and MSE by the 3 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 5-fold validation on the training set. Specifically, the training set has been divided in 5 different subsets (folds) and each hyperparameter configuration was tested on each fold after being trained on the remaining 4 other. The configuration that achieved the highest average R2 score over the 5 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 \( \gamma \) for the classical and RKS-based algorithm, which was set to the value of 1. The number of components in the RKS algorithm was set to the same number of epochs \( E \) of the parallel AQKS. All the algorithms have been implemented using the python library scikit-learn [37]. The hyperparameter spaces for the learning algorithms were:

- **SVR**: \( C : [2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 1, 2, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8] \), \( \epsilon : [10^{-3}, 10^{-2}, 10^{-1}] \)
- **GPR**: \( \beta : [10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}] \)

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 2 settings 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 3 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

---

**Algorithm 2** Proposed implementation parallel AQKS implementation

**Input:** training samples \( \{x_1, \ldots, x_N\} \), probability distributions \( p(A), p(b), E, q \), number of samples to run in each annealing cycle \( \text{samples} \_\text{per} \_\text{run} \)

**Define** \( \text{num} \_\text{iterations} \triangleq N / \text{samples} \_\text{per} \_\text{run} \)

**For** \( i = 1, \ldots, \text{num} \_\text{iterations} \) do

**Initialize** \( Q_i \) as an empty QUBO problem

**For** \( n = 1, \ldots, \text{samples} \_\text{per} \_\text{run} \)

**Define** \( c \triangleq \text{samples} \_\text{per} \_\text{run} \times (i - 1) + n \)

**For** \( e = 1, \ldots, E \)

- **Calculate** random vector \( h_e^c = A_e x_c + b_c \)
- **Add** to the QUBO \( Q_i \) the variables \( \{a_{E_{e(n-1)+c}}, \ldots, a_{E_{e(n-1)+c+q}}\} \)
- **Set** the linear coefficients for \( h_{c,1}, \ldots, h_{c,q} \) respectively
- **Set** the quadratic coefficients \( Q_{E_{e(n-1)+c+j}, E_{e(n-1)+c+k}} = h_{c,j} h_{c,k} \) for \( k, j \in \{0, \ldots, q\} \)

**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 \( \text{samples} \_\text{per} \_\text{run} \times E \times q \) vector by a factor \( 1/E \)

Obtain the transformed feature vectors \( z_{\text{samples} \_\text{per} \_\text{run} \times 1+n} \) of size \( E \times q \) by considering the elements \( \{E \times (n-1) \times q + 1, \ldots, E \times (n) \times q\} \)

**End**

---

**NOMAD** [35] (NASA bio-Optical Marine Algorithm Data set) The second dataset used is also a in-situ dataset and contains several biophysical data information such as surface irradiances, water-leaving radiances, diffuse downwelling attenuation coefficients and chlorophyll concentration values. In this work data taken at 5 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.

For the training phase 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 2 regression methods (SVR, GPR) implemented with the parallel AQKS kernel approximation were tested on 10 different randomly sampled training and test sets of size 200 each. On each of these runs 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 3 different kernel implementation were then compared.
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 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. CONCLUSIONS

The objective of this work was to develop a AQKS kernel approximation implementation on a quantum annealer using parallel quantum annealing 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 quantum annealing 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.

ACKNOWLEDGMENT

The authors gratefully acknowledge support from 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. This work is co-financed by the EUROCC2 project funded by the European High-Performance Computing Joint Undertaking (JU) and EU/EEA states under grant agreement No 101101903. This work is also part of the Center of Excellence (CoE) Research on AI- and Simulation-Based Engineering at Exascale (RAISE) receiving funding from EU’s Horizon 2020 Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733.

REFERENCES

### TABLE I

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

<table>
<thead>
<tr>
<th>Experimental run</th>
<th>AOKS MSE</th>
<th>RKS MSE</th>
<th>Classical MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.38423</td>
<td>0.8451</td>
<td>0.5769</td>
</tr>
<tr>
<td>2</td>
<td>0.8467</td>
<td>0.8565</td>
<td>0.8666</td>
</tr>
<tr>
<td>3</td>
<td>0.522</td>
<td>0.8305</td>
<td>0.8351</td>
</tr>
<tr>
<td>4</td>
<td>0.3851</td>
<td>0.8628</td>
<td>0.8737</td>
</tr>
<tr>
<td>5</td>
<td>0.8183</td>
<td>0.8278</td>
<td>0.8682</td>
</tr>
<tr>
<td>6</td>
<td>0.841</td>
<td>0.8296</td>
<td>0.8331</td>
</tr>
<tr>
<td>7</td>
<td>0.8752</td>
<td>0.8594</td>
<td>0.8503</td>
</tr>
<tr>
<td>8</td>
<td>0.8496</td>
<td>0.8667</td>
<td>0.8852</td>
</tr>
<tr>
<td>9</td>
<td>0.8214</td>
<td>0.8421</td>
<td>0.844</td>
</tr>
<tr>
<td>10</td>
<td>0.8185</td>
<td>0.8605</td>
<td>0.8709</td>
</tr>
</tbody>
</table>

Average: 0.84461

### TABLE II

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

<table>
<thead>
<tr>
<th>Experimental run</th>
<th>AOKS MSE</th>
<th>RKS MSE</th>
<th>Classical MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4094</td>
<td>0.573</td>
<td>0.3067</td>
</tr>
<tr>
<td>2</td>
<td>0.3639</td>
<td>0.4158</td>
<td>0.2232</td>
</tr>
<tr>
<td>3</td>
<td>0.5114</td>
<td>0.6572</td>
<td>0.5599</td>
</tr>
<tr>
<td>4</td>
<td>0.4678</td>
<td>0.4515</td>
<td>0.5191</td>
</tr>
<tr>
<td>5</td>
<td>0.5856</td>
<td>0.6621</td>
<td>0.3698</td>
</tr>
<tr>
<td>6</td>
<td>0.5017</td>
<td>0.5769</td>
<td>0.413</td>
</tr>
<tr>
<td>7</td>
<td>0.6584</td>
<td>0.7147</td>
<td>0.4417</td>
</tr>
<tr>
<td>8</td>
<td>0.65</td>
<td>0.6482</td>
<td>0.5827</td>
</tr>
<tr>
<td>9</td>
<td>0.2984</td>
<td>0.4916</td>
<td>0.3528</td>
</tr>
<tr>
<td>10</td>
<td>0.5722</td>
<td>0.6509</td>
<td>0.5917</td>
</tr>
</tbody>
</table>

Average: 0.50788

---


<table>
<thead>
<tr>
<th>Experimental run</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
<th>SVR</th>
<th>GPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8984</td>
<td>0.8877</td>
<td>0.9091</td>
<td>0.9034</td>
<td>0.912</td>
<td>0.9131</td>
<td>0.1987</td>
<td>0.2178</td>
<td>0.1777</td>
<td>0.189</td>
<td>0.1721</td>
<td>0.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.9039</td>
<td>0.8942</td>
<td>0.91</td>
<td>0.9107</td>
<td>0.9172</td>
<td>0.9294</td>
<td>0.2188</td>
<td>0.2408</td>
<td>0.2049</td>
<td>0.2033</td>
<td>0.1886</td>
<td>0.171</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.888</td>
<td>0.8953</td>
<td>0.8849</td>
<td>0.8877</td>
<td>0.8901</td>
<td>0.8967</td>
<td>0.2575</td>
<td>0.2409</td>
<td>0.2648</td>
<td>0.2582</td>
<td>0.2527</td>
<td>0.2376</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.8975</td>
<td>0.8922</td>
<td>0.8732</td>
<td>0.8346</td>
<td>0.8462</td>
<td>0.8594</td>
<td>0.2184</td>
<td>0.2296</td>
<td>0.2701</td>
<td>0.3522</td>
<td>0.3275</td>
<td>0.2994</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.8788</td>
<td>0.8818</td>
<td>0.8429</td>
<td>0.8163</td>
<td>0.8782</td>
<td>0.8989</td>
<td>0.2488</td>
<td>0.2426</td>
<td>0.3225</td>
<td>0.3771</td>
<td>0.2501</td>
<td>0.2076</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.888</td>
<td>0.8857</td>
<td>0.8826</td>
<td>0.8986</td>
<td>0.9204</td>
<td>0.9227</td>
<td>0.2289</td>
<td>0.2356</td>
<td>0.2399</td>
<td>0.2072</td>
<td>0.1627</td>
<td>0.158</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.8734</td>
<td>0.8757</td>
<td>0.8588</td>
<td>0.888</td>
<td>0.8897</td>
<td>0.8861</td>
<td>0.1967</td>
<td>0.2145</td>
<td>0.2151</td>
<td>0.2121</td>
<td>0.2079</td>
<td>0.2146</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.8903</td>
<td>0.8775</td>
<td>0.8982</td>
<td>0.898</td>
<td>0.9965</td>
<td>0.9082</td>
<td>0.2062</td>
<td>0.2302</td>
<td>0.1914</td>
<td>0.1917</td>
<td>0.1758</td>
<td>0.1726</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.8551</td>
<td>0.8918</td>
<td>0.9019</td>
<td>0.8244</td>
<td>0.8974</td>
<td>0.8958</td>
<td>0.3369</td>
<td>0.2516</td>
<td>0.2281</td>
<td>0.4083</td>
<td>0.2385</td>
<td>0.2423</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.8829</td>
<td>0.8676</td>
<td>0.877</td>
<td>0.8653</td>
<td>0.8403</td>
<td>0.8787</td>
<td>0.2453</td>
<td>0.2773</td>
<td>0.2578</td>
<td>0.2823</td>
<td>0.3347</td>
<td>0.2542</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.88573</td>
<td>0.88425</td>
<td>0.88656</td>
<td>0.8727</td>
<td>0.8898</td>
<td>0.89845</td>
<td>0.23962</td>
<td>0.24137</td>
<td>0.23723</td>
<td>0.26803</td>
<td>0.23106</td>
<td>0.21273</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Eduardo Pasetto** received the B.Sc. degree and M.Sc. degree in Information and Communication Engineering from the University of Trento in 2019 and 2021, respectively. He is currently doing a PhD at Forschungszentrum Jülich and RWTH Aachen University. His main research interest is the application of hybrid quantum-classical Machine Learning frameworks to RS applications.

**Morris Riedel** (Member, IEEE) received his PhD from the Karlsruhe Institute of Technology (KIT) and worked in data-intensive parallel and distributed systems since 2004. He is currently a Full Professor of High-Performance Computing with an emphasis on Parallel and Scalable Machine Learning at the School of Natural Sciences and Engineering of the University of Iceland. Since 2004, Prof. Dr. - Ing. Morris Riedel held various positions at the Juelich Supercomputing Centre of Forschungszentrum Jülich in 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 research interests include high-performance computing, remote sensing applications, pattern recognition, image processing, and data sciences, and he has authored extensively in those fields. Prof. Dr. - Ing. Morris Riedel 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.

**Gabriele Cavallaro** (Member, IEEE) received his B.Sc. and M.Sc. degrees in Telecommunications Engineering from the University of Trento, Italy, in 2011 and 2013, respectively, and a Ph.D. degree in Electrical and Computer Engineering from the University of Iceland, Iceland, in 2016. From 2016 to 2021 he has been the deputy head of the “High Productivity Data Processing” (HPDP) research group at the Jülich Supercomputing Centre, Germany. From 2019 to 2021 he gave lectures on scalable machine learning for remote sensing big data at the Institute of Geodesy and Geoinformation, University of Bonn, Germany. Since 2022, he is the Head of the “AI and ML for Remote Sensing” Simulation and Data Lab at the Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany and an Adjunct Associate Professor with the School of Natural Sciences and Engineering, University of Iceland, Iceland. He is also the Chair of the High-Performance and Disruptive Computing in Remote Sensing (HDCRS) Working Group of the IEEE GRSS ESI Technical Committee and a Visiting Professor at the Φ-lab of the European Space Agency (ESA) in the context of the Quantum Computing for Earth Observation (QC4EO) initiative. Since October 2022 he serves as an Associate Editor of the IEEE Transactions on Image Processing (TIP). He also serves on the scientific committees of several international conferences and he is a referee for numerous international journals. He 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). His research interests cover remote sensing data processing with parallel machine learning algorithms that scale on distributed computing systems and cutting-edge computing technologies, including quantum computers.

**Kristel Michielsen** received her PhD from the University of Groningen, the Netherlands, for work on the simulation of strongly correlated electron systems in 1993. Since 2009 she is group leader of the research group Quantum Information Processing at the Jülich Supercomputing Centre, Forschungszentrum Jülich (Germany) and is also Professor of Quantum Information Processing at RWTH Aachen University (Germany). Her current research interests include quantum computation, quantum annealing, quantum statistical physics, event-based simulation methods of quantum phenomena, logical inference approach to quantum mechanics and computational electrodynamics.