SoftwareX 26 (2024) 101756

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

EVOVAQ: EVOlutionary algorithms-based toolbox for VAriational Quantum circuits

Giovanni Acampora [a], Carlos Cano Gutiérrez [b], Angela Chiatto [a], José Manuel Soto Hidalgo [c], Autilia Vitiello [a,*]

[a] Department of Physics "Ettore Pancini", University of Naples Federico II, Naples, Italy
[b] Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
[c] Department of Computer Engineering, Automatics and Robotics, University of Granada, Granada, Spain

ARTICLE INFO

Keywords:
Evolutionary algorithms
Python package
Quantum computing
Quantum machine learning
Variational quantum algorithms

ABSTRACT

Evolutionary Algorithms (EAs) are becoming increasingly popular for training Variational Quantum Circuits (VQCs) due to their ability to conserve quantum resources. However, there is currently a lack of user-friendly tools for implementing this approach. To address this issue, this paper proposes EVOVAQ, a Python-based framework designed to simplify the use of EAs for training VQCs. EVOVAQ seamlessly integrates evolutionary computation with quantum libraries such as Qiskit, making it easy to use for both quantum computing and EAs communities. Furthermore, EVOVAQ's scalability enables the development of customized solutions, promoting innovation in the quantum computing field.

Code metadata

Current code version	v1.0.21
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00132
Permanent link to Reproducible Capsule	
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python≥3.9 numpy == 1.23.5 tabulate == 0.8.10 tqdm == 4.64.1 matplotlib == 3.5.1 pandas == 1.4.2 openpyxl == 3.0.9
If available Link to developer documentation/manual	https://evovaq.readthedocs.io/en/latest/index.html
Support email for questions	angela.chiatto@unina.it

1. Motivation and significance

Quantum computing stands at the forefront of the oncoming technological revolution, thus attracting investments from both private and public sectors for its wide-ranging potential impacts. The shared goal of realizing short-term benefits has fostered closer collaborations among governments, companies, and academic institutions, nurturing the growth of the so-called quantum ecosystem. Within this expanding community, efforts are concentrated on demonstrating the practical supremacy of current-generation quantum computers over digital computers, a concept known as quantum advantage, particularly in tasks like optimization and machine learning [1].

The current and forthcoming era of quantum computing, known as Noisy Intermediate-Scale Quantum (NISQ) [2], is characterized by constraints such as the limited number of physical qubits available on

* Corresponding author.
E-mail address: autilia.vitiello@unina.it (Autilia Vitiello).

https://doi.org/10.1016/j.softx.2024.101756
Received 29 February 2024; Received in revised form 24 April 2024; Accepted 29 April 2024
Available online 9 May 2024

quantum processors and the presence of noise affecting the reliability of extended sequences of quantum operations. As a result, existing quantum resources are insufficient for constructing fault-tolerant large quantum processors capable of executing pivotal algorithms like Shor's for prime factorization [3] and Grover's for database search [4]. In this scenario, Variational Quantum Circuits (VQCs) have gained prominence as an approach to quantum algorithms in the NISQ era. VQCs are quantum algorithms that harness a quantum computer, trained by a classical counterpart, to perform diverse tasks. Essentially, a task is encoded into a parameterized cost function, evaluated using a quantum computer, and fine-tuned by a classical optimizer [5]. Therefore, VQCs emerge as hybrid quantum–classical algorithms requiring fewer quantum resources, such as qubit count and circuit depth, compared to algorithms tailored for fault-tolerant devices like Shor's and Grover's. Moreover, this versatile approach has proven effective in solving problems across various research domains, spanning from physics to chemistry and from optimization to machine learning. Nevertheless, challenges remain in providing ready-to-implement quantum algorithms, as some critical issues, especially related to trainability, limit the performance of VQCs in large-scale applications.

The main problem for VQCs is the emergence of barren plateau landscapes with narrow gorges as the number of qubits increases [6,7]. This phenomenon is due to the fact that the gradients of the cost function vanish exponentially in the number of qubits, thus leading to a flat landscape. This implies that training VQCs can be particularly challenging, especially when employing conventional gradient-based optimization algorithms. Furthermore, the noise that plagues NISQ devices may have an impact on the trainability and accuracy of VQCs: in fact, it could distort the landscape so that the noisy global optimum no longer corresponds to the noise-free global optimum, and it could affect the final value of the optimal cost [8]. As a consequence, there is a strong need of computational methodologies aimed at addressing these issues in VQCs, always attempting to guarantee maximum efficiency in terms of quantum resource requirements as well.

Recently, EAs are proving to be the most suitable solution to alleviate some of the issues previously described. EAs are indeed efficient heuristic search methods with powerful characteristics of robustness and flexibility which have already shown good performance on training VQCs, as we will shortly outline. EAs require no fitness gradient information of any kind to proceed, they are easy to process in parallel due to their population-based nature, and have the ability to escape from local minima where deterministic optimization methods may fail or are not applicable. Specifically, in [9,10], Genetic Algorithms (GAs) have been introduced as classical optimizers to train VQCs in combinatorial optimization and classification tasks respectively. Moreover, for classification tasks, the performance of GA in visiting the problem search space has been improved in [11] by combining it with a local search technique, thus resulting in a Memetic Algorithm (MA). In [12], the first comparative study between real-coded evolutionary algorithms, such as Differential Evolution (DE), Particle Swarm Optimization (PSO) and GA, has been carried out to investigate the performance of these optimization methodologies in training VQCs used as classifiers for well-known benchmark datasets. In [13], DE has been applied and compared to other state-of-art optimizers for VQC training in physics. PSO has been proposed for VQC training in chemistry [14], combinatorial optimization [15], and data-driven methods for generative modelling of classical data [16]. Three variants of Estimation of Distribution Algorithms (EDA) have been applied and compared to other standard optimizers to train VQCs to solve Max Cut problem and simulate the behaviour of a molecule [17]. Evolutionary optimization has also been shown to reach similar or even superior performance compared to gradient-based Quantum Reinforcement Learning problems [18]. Moreover, EAs have shown promising results for automatically identifying appropriate circuit architectures for VQCs. To this end, a genome-length-adjustable evolutionary algorithm has been proposed in [19] to design a robust parameterized quantum circuit, which is optimized over variations of both circuit architecture and gate parameters. In [20], an evolutionary quantum neural architecture search has been proposed for image classification using quantum neural networks.

Despite their increasing popularity for accomplishing the training of VQCs, current quantum frameworks and platforms do not offer implementations of EAs for VQC training. Certainly, several Python packages for evolutionary computation are available in the literature. The most popular is DEAP (Distributed Evolutionary Algorithms in Python), which provides a flexible framework for building and testing EAs [21]. Other available Python packages that offer implementations of specific EAs are EvoStrat implementing the evolution strategy (ES) algorithm [22], and CMA-ES implementing the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [23]. However, these Packages do not provide an integration of EA to solve specific tasks. Other Python packages were designed to solve specific tasks by means of EAs, such as PyGAD (Python Genetic Algorithm Library) for building genetic algorithms and optimizing machine learning algorithms [24], BluePyOpt for parameter optimization of neural models using EAs [25], EC-KitY for doing evolutionary computation in a seamless integration with scikit-learn for machine learning [26]. However, there is a lack of Python package to tackle tasks in the field of VQCs using EAs. Only Qiskit from IBM provides the implementation of a single EA for VQC training, specifically the Continuous Univariate Marginal Distribution Algorithm (UMDA) [27]. Consequently, the traditional EAs mentioned above remain unavailable within these frameworks, which prevents them from becoming routinely used for training VQCs.

With this in mind, this paper introduces EVOVAQ, a novel evolutionary computation-based framework dedicated to VQCs, which offers a user-friendly interface to existing environments for implementing quantum circuits, and access to ready-to-use EAs proposed in the literature. The goal of EVOVAQ is to facilitate seamless the integration between evolutionary computation and quantum libraries like Qiskit, while ensuring ease of use, for both quantum computing and EAs communities. Additionally, EVOVAQ incorporates the implementation of algorithms, such as MAs, which may not be readily accessible in the available evolutionary libraries. In the next section, the features of this novel framework will be discussed in detail.

2. Software description

EVOVAQ is a novel Python framework designed to easily train VQCs through EAs to perform different tasks such as optimization and classification. It has been written in Python in order to be easily integrated with the most popular quantum computing libraries such as Qiskit[1] from IBM, Circq[2] from Google, PyQuil[3] from Rigetti that are in fact developed in Python programming language.

EVOVAQ is included in the official Python Package Index (PyPI). Therefore, in order to install it, it is possible to simply run the following shell command:

```
pip install evovaq
```

which will handle all dependencies automatically and always install the latest version. Hereafter, the main architectural components of EVOVAQ and its functionalities are given in detail.

2.1. Software architecture

Fig. 1 shows a UML class diagram aimed at highlighting the three main components of the EVOVAQ's architecture. To start, the `Problem` class is specially designed to instantiate the problem at issue. Precisely, the training of VQCs consists of solving real single-objective problems.

[1] Qiskit: https://www.ibm.com/quantum/qiskit.
[2] Circq: https://quantumai.google/cirq.
[3] PyQuil: https://www.rigetti.com/applications/pyquil.

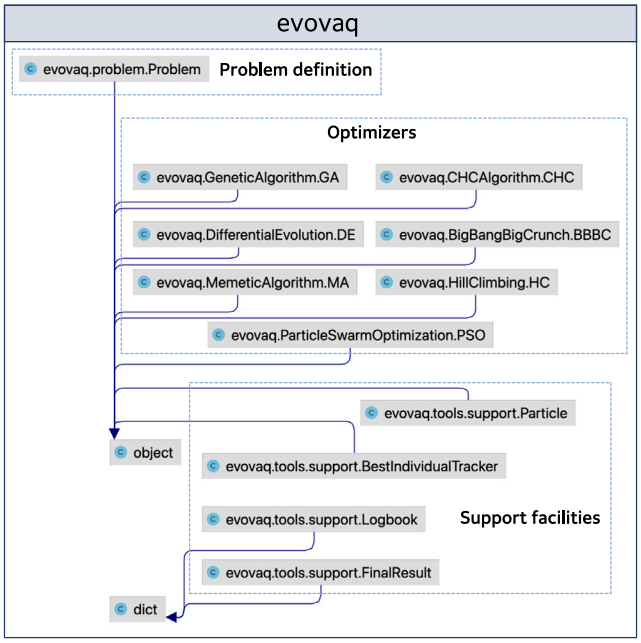

Fig. 1. UML class diagram of EVOVAQ. The three main components are highlighted: **Problem definition**, **Optimizers**, and **Support facilities**.

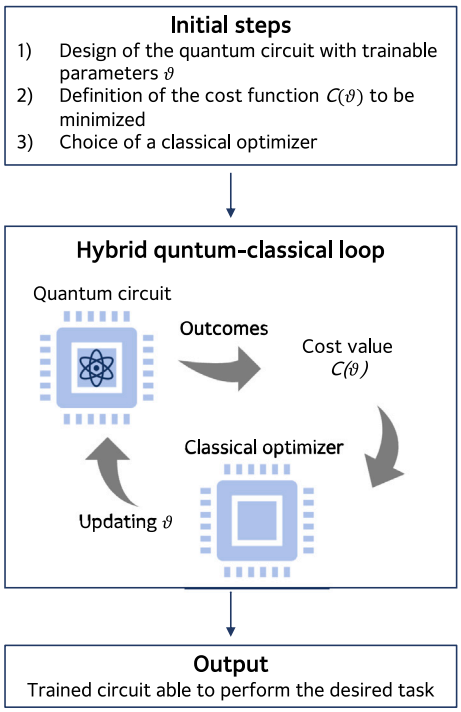

Fig. 2. Graphical representation of the general workflow of VQCs.

Thus, EVOVAQ allows the definition of this kind of optimization problems by just requiring few information, such as the number of real circuit parameters to be optimized, their upper and lower bounds, and the cost function to be minimized. Hence, the `Problem` instance is then used during all the optimization process to generate possible solutions, evaluate the fitness function and check the parameter bounds. In particular, the cost function to be defined involves a quantum circuit – implemented, for instance, using Qiskit – that relies on real parameters to be optimized for a desired task. Therefore, a possible solution is a NumPy array [28] of these real parameters to be bound to the quantum circuit, with the resultant cost or also fitness value derived from measuring the circuit.

The `Optimizers` are classes that the user can use to solve the optimization problem at issue, that it, learning the real parameters of a quantum circuit. EVOVAQ makes available several evolutionary algorithms ready to be used as optimizers for training quantum circuits. The user can select among some evolutionary algorithms already applied in this context in literature, including **Genetic Algorithms**, **Differential Evolution**, **Particle Swarm Optimization** and **Memetic Algorithms** obtained by combining GA and **Hill Climbing (HC)**. Furthermore, it is also possible to employ dithering in DE, and readily combine DE and HC to form MA. Moreover, EVOVAQ includes other two techniques, which seem to have promising features for training VQCs, namely **Big Bang Big Crunch (BBBC)** and **Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation (CHC)** algorithm. Both techniques have proven to be very successful in many real-world applications, so EVOVAQ allows us to start testing them in the quantum domain as well. An optimizer can be instantiated by setting symbolic and numerical hyper-parameter values. To facilitate non-experts, default hyper-parameters values have been set. Several ready-to-use genetic operators are available in the `operators` module of the `tools` directory. However, it is possible to integrate user-built operators into this framework. With regard to the instantiation of MA, user-built local and/or global search methods can be integrated as well.

Finally, in the module `support`, we have developed all the functions and classes that serve as support facilities for the optimizers. The main classes are: `BestIndividualTracker`, which keeps track of the best solution ever found during the search; `Logbook`, which stores the statistics of fitness values during the search; `FinalResult`, which collects the information of the final results.

An in-depth description of all functions, classes, return types, arguments and other is reported in the API guide available in the documentation.

2.2. Software functionalities

The main functionality of the current EVOVAQ version is to offer a user-friendly interface between the quantum libraries implementing VQCs and EAs used for their training in specific tasks. This implies that, given the versatility of the VQC workflow, a wide range of tasks, spanning from classification to optimization, can be accomplished by training VQCs with EAs using EVOVAQ. Basically, regardless of the task at hand, as depicted in Fig. 2, the general workflow of VQC involves the following three initial steps: (1) designing a quantum circuit with trainable real parameters, known as the ansatz; (2) defining a cost function tailored to the desired task; (3) selecting a classical optimizer to train the ansatz. By assigning some initial values to the free parameters of the ansatz, the quantum circuit's measurements are utilized to compute the cost function, while the optimization algorithm, running on a classical computer, is employed to update these parameters to minimize the cost function. This iterative quantum–classical hybrid loop concludes once a specified stopping criterion is met. Subsequently, the trained circuit is capable of executing the desired task. In order to better understand the functionality of EVOVAQ in this workflow, Fig. 3 provides an overview of the interaction between EVOVAQ and possible users. In detail, after the definition of the problem and the choice of a classical optimizer, the user can train VQCs by just applying the method `optimize` of the algorithm instance. The classical optimizer operates by assuming a default value for the hyper-parameters, such as the population size and the stopping criterion, that is, the maximum number of fitness evaluations or generations. Alternatively, the user may specify a different value for such hyper-parameters. Finally, the optimization procedure returns a result containing the best parameters of the quantum circuit to be learned.

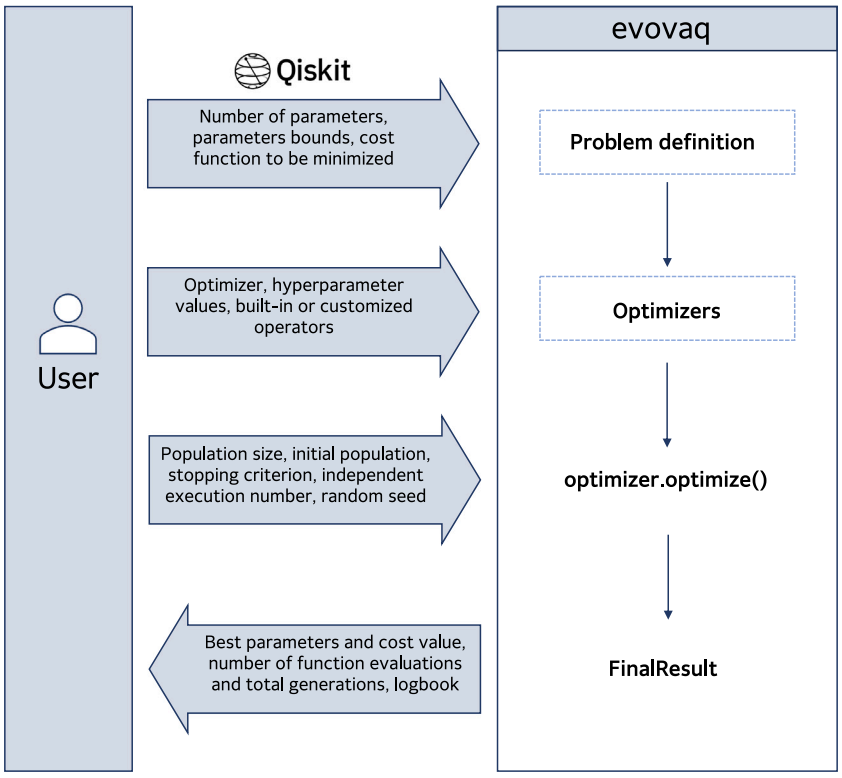

Fig. 3. Overview of EVOVAQ's interactions with the user.

3. Illustrative examples

EVOVAQ can be practically tested on two different application domains, including optimization and machine learning. In the documentation, there are two tutorials: (1) how to train quantum classifiers, implemented in Qiskit, through MA obtained from the combination of GA and HC; (2) how to solve the Maximum Cut problem on a graph using QAOA circuit implemented in Qiskit and trained by PSO. In order to deliver a comprehensive discussion of the tutorials presented in the documentation, Appendix provides the essentials of quantum computing and an in-depth description of VQCs in machine learning and optimization.

This section, instead, describes all the steps in the first tutorial illustrating how EVOVAQ works in the context of machine learning. For the sake of simplicity, the classification problem for the first two classes of the Iris dataset is addressed. In addition to Qiskit for the implementation of quantum circuits, scikit-learn [29], the traditional Python library employed by machine learning practitioners, is used to load the Iris data, a pre-processing method and some metrics. The following modules are then imported:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import log_loss
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from qiskit.circuit.library import
    ZZFeatureMap, RealAmplitudes
from qiskit import Aer, execute
from evovaq.problem import Problem
from evovaq.GeneticAlgorithm import GA
from evovaq.HillClimbing import HC
from evovaq.MemeticAlgorithm import MA
import evovaq.tools.operators as op
import numpy as np
```

Initially, the process involves loading classical data and treating it as a conventional classification problem. The data is then partitioned into training and testing sets, followed by a pre-processing technique, as follows:

```
iris = load_iris()

# For the sake of simplicity, we consider all
    the four features but only two classes
iris_data = iris.data[:100, :4]
iris_target = iris.target[:100]    # 0 or 1

# Split into train and test subsets
train_data, test_data, train_labels,
    test_labels = train_test_split(iris_data,
    iris_target, test_size=0.2, random_state=42)

# Pre-process data
scaler = MinMaxScaler()
scaler.fit(train_data)

train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)
```

The classification model is here a parameterized quantum circuit, which is trained by MA on a classical processor. In particular, given their widespread use in the Quantum Machine Learning community, the ZZ Feature Map for encoding the classical data in quantum states and the Real Amplitude as ansatz are specified to implement the quantum classifier as follows:

```
# Encode classical data in a quantum system
    through a FeatureMap
dim = train_data.shape[1]
feature_map = ZZFeatureMap(dim, reps=1,
    entanglement="linear")

# Define an Ansatz to be trained
```

```
ansatz = RealAmplitudes(num_qubits=dim, reps=1,
    entanglement="circular")

# Put together our quantum classifier
circuit = feature_map.compose(ansatz)

# Measure all the qubits to retrieve label
    information
circuit.measure_all()
```

To get the label of an instance, it is necessary to bind the feature and parameter values to the quantum circuit, and later measure the final state. From the measurement procedure, the probability distribution of measuring the possible 2^n bit-strings is determined. This distribution is used to compute the probability of reading label 0 or 1. Specifically, the widely used parity mapping is adopted: for each measured bit-string, the count of 1's is determined, and if this count is odd, its probability contributes to the probability of class 1, otherwise of class 0, as defined as follows:

```
def get_label_prediction(circuit, features,
    params):
    # Bind the parameters to our quantum
        classifier
    bound_circuit =
        circuit.bind_parameters(np.concatenate(
    (features, params)))
    backend = Aer.get_backend("qasm_simulator")
    counts = execute(bound_circuit,
        backend).result().get_counts()

    # Read the label by considering the parity
        mapping of the final quantum state
    parity_1 = 0
    for state, count in counts.items():
        if state.count("1")    % 2 == 1:
                    parity_1 += count
    return parity_1 / sum(counts.values())
```

The resulting outputs from the training instances can be used to calculate the cross-entropy to be minimized, coinciding with the cost value of the solution considered, defined as follows:

```
def cost_function(params):
    predictions =
        [get_label_prediction(circuit,
        features, params) for features in
        train_data]
    return log_loss(train_labels, predictions)
```

The minimization problem to be solved can be now easily defined by instantiating the `Problem` class of EVOVAQ, as follows:

```
problem = Problem(ansatz.num_parameters,
    ansatz.parameter_bounds, cost_function)
```

The classical optimizer used to train the ansatz parameters is MA which is composed of a global and local search method, GA and stochastic variant of HC respectively in this example. Of course, HC requires the definition of a strategy indicating how to create a neighbour of a possible solution to the problem at hand. The following code shows how to easily implement MA in EVOVAQ.

```
# Define the global search method
global_search = GA(selection=op.sel_tournament,
    crossover=op.cx_uniform,
    mutation=op.mut_gaussian, sigma=0.2,
    mut_indpb=0.15,
                cxpb=0.9, tournsize=5)

# Create a neighbour of a possible solution
def get_neighbour(problem, current_solution):
    neighbour = current_solution.copy()
    index = np.random.randint(0,
        len(current_solution))
    _min, _max = problem.param_bounds[0]
    neighbour[index] = np.random.uniform(_min,
        _max)
    return neighbour

# Define the local search method
local_search =
    HC(generate_neighbour=get_neighbour)

# Compose the global and local search method
    for a Memetic Algorithm
optimizer =
    MA(global_search=global_search.evolve_
population, sel_for_refinement=op.sel_best,
    local_search=local_search.stochastic_var,
    frequency=0.1, intensity=10)
```

The training is performed by applying the `optimize` method to the selected optimizer as follows:

```
res = optimizer.optimize(problem, 10,
    max_gen=10, verbose=True, seed=42)
```

Therein the user can provide the population size, the maximum number of generations to finish, the random seed and also the verbose option to enable the print the statistics of the fitness values during evolution. Specifically, when verbose is `True`, the number of fitness values completed, the minimum, maximum, mean and standard deviation of the fitness values in the population in the ith generation, as well as the progress bar showing the percentage of the completed generations are printed. The final result contains the best parameters and cost values, the total number of fitness evaluations and generations, and the logbook containing the information collected during the evolution.

The trained model is now ready to be used to predict labels for the test subset. The quality of the trained model is evaluated by computing the accuracy value as follows:

```
test_predictions = [1 if
    get_label_prediction(circuit, features,
    res.x) > 0.5 else 0 for features in
    test_data]
test_accuracy = accuracy_score(test_labels,
    test_predictions)
print("Accuracy on the test subset:",
    test_accuracy)
```

4. Impact

The growing quantum community looks at VQCs as promising candidates to achieve the quantum advantage in key application scenarios, from optimization to machine learning, from chemistry to physics. Despite their potential benefits, the performance of VQCs is still limited by critical challenges such as trainability, efficiency, and resulting accuracy [5]. EAs are emerging as a suitable strategy to address these challenges [9–20]. Hence, the great impact of EVOVAQ that makes possible and simple the application of EAs to train VQCs by assuming a key role in the development of quantum computing area. Moreover, EVOVAQ design allows for the customization and integration of operators, enhancing the workflow's flexibility. To conclude, EVOVAQ is not only a pragmatic tool enabling the construction of bespoke experiments but also a catalyst in advancing research towards demonstrating quantum advantage through its ready-to-implement quantum algorithms.

5. Conclusions

EVOVAQ is a user-friendly Python package developed to enable the use of EAs for training VQCs in a simple interface with existing quantum libraries. To motivate the importance of this tool, we discussed

why evolutionary techniques are promising classical optimizers for the practical implementation of VQCs, and thus for the demonstration of near-term quantum advantage. By means of EVOVAQ, extensive experimentation can be conducted to investigate which EAs are best suited for training VQCs in specific tasks.

In the future, EVOVAQ will be extended by introducing the implementation of other more innovative EAs. Additionally, EVOVAQ's capabilities will be enhanced to address circuit design by optimizing also ansatz of the quantum circuits. In this way, EVOVAQ will address various challenges in the context of VQCs, establishing itself as an efficient tool for practical implementation of VQCs.

CRediT authorship contribution statement

Giovanni Acampora: Writing – review & editing, Supervision, Methodology, Conceptualization. **Carlos Cano Gutiérrez:** Writing – review & editing, Supervision. **Angela Chiatto:** Writing – original draft, Visualization, Software, Investigation, Conceptualization. **José Manuel Soto Hidalgo:** Writing – review & editing, Supervision. **Autilia Vitiello:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Angela Chiatto reports financial support was provided by Institute of Electrical and Electronics Engineers. Carlos Cano reports financial support was provided by MCIN/AEI/10.13039/501100011033/FEDER. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This project was supported by the IEEE Computational Intelligence Society Graduate Student Research Grant, won by the author Angela Chiatto in 2023. CC was supported by PID2021-128970OA-I00 funded by MCIN/AEI/10.13039/501100011033/FEDER.

Appendix

This Appendix provides the basic concepts of quantum computing, and a more detailed description of variational quantum circuits in machine learning and optimization contexts.

Basic concepts of quantum computing

The fundamental units of a quantum computer are the *qubits*, *i.e.* the quantum version of digital bits [30]. Mathematically, the most general qubit state $|q\rangle$ is expressed in bra-ket notation as a combination of both 0 and 1 logic states, namely $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, where $\{|0\rangle, |1\rangle\}$ is the *computational basis*, α_0 and α_1 are complex numbers denoting the *probability amplitudes*. Upon a measurement, the qubit $|q\rangle$ collapses to either $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha_0|^2$ and $|\alpha_1|^2$, respectively. These amplitudes must satisfy the normalization condition, hence $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

A quantum computer comprises a set of n qubits, indicated as *n-qubit register*, manipulated to perform computations. The general state $|q_n\rangle$ of n-qubit register is written in bra-ket notation as $|q_n\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$, where each α_x is the probability amplitude related to the n-bit string x, therefore $\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$. The logical operations on one or more qubits are defined by the *quantum logic gates*. They are the unitary operators represented by unitary matrices relative to the computational basis. For instance, the well-known Pauli unitary matrices (X, Y, Z) implement single-qubit gates. The exponentiation of Pauli gates defines other examples of single-qubit operators, known as *rotation gates*, whose actions depend on real tunable parameters $(\theta_X, \theta_Y, \theta_Z)$ as follows:

$$R_X(\theta_X) = e^{-i\frac{\theta_X}{2}X}, \quad R_Y(\theta_Y) = e^{-i\frac{\theta_Y}{2}Y}, \quad R_Z(\theta_Z) = e^{-i\frac{\theta_Z}{2}Z}. \tag{1}$$

The prototypical multi-qubit gate is the controlled version of X gate, called CNOT. This two-qubit gate is made up of a control and a target qubit: X gate is applied to the target qubit, only if the control qubit is in the state $|1\rangle$.

A *quantum algorithm* is delineated by a finite sequence of quantum logic gates, forming a *quantum circuit*, alongside a final measurement operation. Given a unitary operation U acting on a quantum system $|q_n\rangle$, the resulting quantum state $|q'_n\rangle$ has the following form:

$$|q'_n\rangle = U|q_n\rangle = \sum_{x\in\{0,1\}^n} \alpha_x U|x\rangle = \sum_{x\in\{0,1\}^n} \alpha'_x |x\rangle, \tag{2}$$

where α'_x, with $x \in \{0,1\}^n$, are different probability amplitudes which affect the measurement outcomes and thus the algorithm result. To obtain the probability distribution of the possible results, the same quantum circuit is executed many times. For VQCs, the unitary operator's behaviour is determined by a specific set of parameter values, denoted as ϑ, hence the resultant quantum state is

$$|q'_n(\vartheta)\rangle = U(\vartheta)|q_n\rangle = \sum_{x\in\{0,1\}^n} \alpha_x U(\vartheta)|x\rangle = \sum_{x\in\{0,1\}^n} \alpha'_x(\vartheta)\ |x\rangle. \tag{3}$$

Consequently, by adjusting the values within the set ϑ, it is becoming possible to search interesting solutions to a given problem.

Variational quantum circuits in machine learning and optimization

The versatility of VQC workflow illustrated in Fig. 2 opens up the possibility of designing different parameterized quantum circuits tailored to a desired task.

In the domain of machine learning, VQCs play a key role as innovative classification models [31], leveraging quantum phenomena such as entanglement and superposition. That involves training a quantum computer on labelled data to derive labels for unseen data instances. Specifically, a variational quantum classifier comprises a feature map, enabling the encoding of classical data in a quantum circuit, and an ansatz, serving as a trainable model. The outcomes obtained from the circuit measurement are utilized to generate labels and compute the value of the cost function to learn the optimal circuit parameters for the given task through a classical optimizer. Formally, a parameterized quantum circuit, represented by the parameterized unitary operator $U(\vartheta)$, is trained on labelled data $\mathcal{D}_{train}$ to generate predictions for unseen data. This quantum model $U(\vartheta)$, or ansatz, has a multi-layered architecture, where each layer typically comprises rotation gates defined in (1) alongside CNOT gates. Given their similarity to the layered structure controlled by learnable parameters of artificial neural networks, VQCs are commonly regarded as *quantum neural networks*. Just like their classical counterparts, these quantum models incorporate an input layer to load classical data for later processing. In particular, the initial layer is represented by an initial quantum state encoding the classical data. The unitary operator U_ϕ, called *feature map*, actually connects the real space to the Hilbert space of quantum states. Given a training dataset $\mathcal{D}_{train} = \{\mathbf{f}_1, \ldots, \mathbf{f}_M\}$, each feature vector $\mathbf{f} \in \mathbb{R}^N$ is therefore encoded in a quantum state by implementing another quantum circuit $U_{\phi(\mathbf{f})}$, now parameterized by the feature vectors of $\mathcal{D}_{train}$. Assuming that the initial n-qubit register is initialized to logical zeros ($|0\ldots0\rangle$), the input layer consists of the following quantum state:

$$|\phi^{in}(\mathbf{f})\rangle = U_{\phi(\mathbf{f})}|0\ldots0\rangle. \tag{4}$$

By composing the circuits implementing the feature map $U_{\phi(\mathbf{f})}$ and the ansatz $U(\vartheta)$, the output quantum state is expressed as follows:

$$\left|\phi^{out}(\vartheta;\mathbf{f})\right\rangle = U(\vartheta)\left|\phi^{in}(\mathbf{f})\right\rangle = U(\vartheta)U_{\phi(\mathbf{f})}\left|0\ldots0\right\rangle \tag{5}$$

A measurement operation on this output state provides the probabilities of retrieving different bit-strings, which can be used to read the label and compute the cost function $C(\vartheta)$ (Cross Entropy or even Mean Squared Error) to train the circuit parameters ϑ.

With respect to the optimization field, QAOA [32] employs the VQC workflow to address combinatorial optimization problems, particularly Quadratic Unconstrained Binary Optimization (QUBO) instances. These instances involve binary variables without constraints and aim to minimize a quadratic objective function. Practically, QAOA tackles the QUBO problem by minimizing the energy of the quantum computer. The optimal bit-string is in fact encoded in the ground state, *i.e.* the lowest energy quantum state, of the ansatz proposed in [32]. This approach results in the final state approximating the ground state that encodes the optimal solution. Formally, QUBO instances involve n binary and unconstrained variables $x \in \{0,1\}^n$, and a quadratic objective function $C(x)$. In the QAOA variational formulation, the cost function $C(x)$ to be minimized is promoted to a problem Hamiltonian H_C, namely an Hermitian operator describing the total energy of a n-qubit register. Specifically, the total energy of a n-qubit register in the computational basis state $|x\rangle$ is represented by the following expectation value of H_C in that basis state:

$$\langle x|\, H_C\, |x\rangle = C(x), \tag{6}$$

which is equivalent to the cost value $C(x)$ associated with the n-bit string x. In this way, the ground state of this n-qubit system $|\psi^*\rangle$, *i.e.* the lowest energy state, encodes the optimal solution x^* to a given problem.

Starting from a well-known ground state $|\psi_0\rangle$ of another Hamiltonian H_0, called *mixer* Hamiltonian, this state can slowly evolve to the desired ground state $|\psi^*\rangle$ of H_C encoding the optimal solution. As demonstrated in [32], this evolution is represented by the parameterized unitary operator $U(\gamma,\beta)$ acting on a n-qubit register. Practically, this transformation is implemented by p repetitions of cost and mixer layers. The cost layer is represented by the unitary operator $U_{H_C}(\gamma_i) = e^{-i\gamma_i H_C}$ parameterized by γ_i, while the mixer layer by the unitary operator $U_{H_0}(\beta_i) = e^{-i\beta_i H_0}$ parameterized by β_i. By altering these layers on the initial ground state $|\psi_0\rangle$, the final quantum state approximating the optimal solution for a given QUBO instance has the following form:

$$|\psi(\gamma,\beta)\rangle = U_{H_0}(\beta_p)U_{H_C}(\gamma_p)\cdots U_{H_0}(\beta_1)U_{H_C}(\gamma_1)\left|\psi_0\right\rangle \tag{7}$$

By measuring all the qubits, it is possible to compute the following expectation value:

$$\langle C(\beta,\gamma)\rangle = \langle\psi(\beta,\gamma)|\, H_C\, |\psi(\beta,\gamma)\rangle\,, \tag{8}$$

where $C(\beta,\gamma)$ represents the cost function to be minimized in this formulation. By decomposing the quantum state $|\psi(\beta,\gamma)\rangle$ in the computational basis states, $|\psi(\beta,\gamma)\rangle = \sum_{x\in\{0,1\}^n}\sqrt{p_x(\beta,\gamma)}\, e^{i\delta_x}\,|x\rangle$, where $\sqrt{p_x(\beta,\gamma)}\, e^{i\delta_x}$ is the exponential form of a complex number, the cost function in (8) can be expressed as:

$$\langle C(\beta,\gamma)\rangle = \sum_{x\in\{0,1\}^n} p_x(\beta,\gamma)C(x) \tag{9}$$

where $p_x(\beta,\gamma)$ is the probability of measuring the corresponding n-bit string x with the values (β,γ), and $C(x)$ is the corresponding cost value. Therefore, finding the solution to a QUBO instance via QAOA means finding the set of the optimal parameters (β^*,γ^*). In this way, the state $|\psi(\beta^*,\gamma^*)\rangle$ approximates the desired ground state $|\psi^*\rangle$ encoding the optimal solution, and the probability of measuring the optimal n-bit string x^* is highest.

References

[1] Intelligence NM. Seeking a quantum advantage for machine learning. Nat Mach Intell 2023;5(8):813. http://dx.doi.org/10.1038/s42256-023-00710-9.

[2] Preskill J. Quantum computing in the NISQ era and beyond. Quantum 2018;2:79.

[3] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review 1999;41(2):303–32.

[4] Grover LK. A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on theory of computing. 1996, p. 212–9.

[5] Cerezo M, Arrasmith A, Babbush R, Benjamin SC, Endo S, Fujii K, et al. Variational quantum algorithms. Nat Rev Phys 2021;3(9):625–44.

[6] McClean JR, Boixo S, Smelyanskiy VN, Babbush R, Neven H. Barren plateaus in quantum neural network training landscapes. Nat Commun 2018;9(1):4812.

[7] Cerezo M, Sone A, Volkoff T, Cincio L, Coles PJ. Cost function dependent barren plateaus in shallow parametrized quantum circuits. Nat Commun 2021;12(1):1791.

[8] Wang S, Fontana E, Cerezo M, Sharma K, Sone A, Cincio L, et al. Noise-induced barren plateaus in variational quantum algorithms. Nat Commun 2021;12(1):6961.

[9] Acampora G, Chiatto A, Vitiello A. Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm. Appl Soft Comput 2023;110296. http://dx.doi.org/10.1016/j.asoc.2023.110296, URL https://www.sciencedirect.com/science/article/pii/S1568494623003149.

[10] Acampora G, Chiatto A, Vitiello A. Training variational quantum circuits through genetic algorithms. In: 2022 IEEE congress on evolutionary computation. IEEE; 2022.

[11] Acampora G, Chiatto A, Vitiello A. Training circuit-based quantum classifiers through memetic algorithms. Pattern Recognit Lett 2023. http://dx.doi.org/10.1016/j.patrec.2023.04.008, URL https://www.sciencedirect.com/science/article/pii/S0167865523001162.

[12] Acampora G, Chiatto A, Vitiello A. A comparison of evolutionary algorithms for training variational quantum classifiers. In: 2023 IEEE congress on evolutionary computation. CEC, 2023, p. 1–8. http://dx.doi.org/10.1109/CEC53210.2023.10254076.

[13] Faílde D, Viqueira JD, Mussa Juane M, Gómez A. Using differential evolution to avoid local minima in variational quantum algorithms. Sci Rep 2023;13(1):16230.

[14] Benedetti M, Garcia-Pintos D, Perdomo O, Leyton-Ortega V, Nam Y, Perdomo-Ortiz A. A generative modeling approach for benchmarking and training shallow quantum circuits. npj Quantum Inf 2019;5(1):45.

[15] Guerreschi GG, Hogaboam J, Baruffa F, Sawaya NP. Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits. Quantum Sci Technol 2020;5(3):034007.

[16] Zhu D, Linke NM, Benedetti M, Landsman KA, Nguyen NH, Alderete CH, et al. Training of quantum circuits on a hybrid quantum computer. Sci Adv 2019;5(10):eaaw9918.

[17] Soloviev VP, Larrañaga P, Bielza C. Variational quantum algorithm parameter tuning with estimation of distribution algorithms. In: 2023 IEEE congress on evolutionary computation. CEC, IEEE; 2023, p. 1–9.

[18] Chen SY-C, Huang C-M, Hsing C-W, Goan H-S, Kao Y-J. Variational quantum reinforcement learning via evolutionary optimization. Mach Learn: Sci Technol 2022;3(1):015025.

[19] Huang Y, Li Q, Hou X, Wu R, Yung M-H, Bayat A, et al. Robust resource-efficient quantum variational ansatz through an evolutionary algorithm. Phys Rev A 2022;105(5):052414.

[20] Li Y, Liu R, Hao X, Shang R, Zhao P, Jiao L. EQNAS: Evolutionary quantum neural architecture search for image classification. Neural Netw 2023;168:471–83.

[21] Fortin F-A, De Rainville F-M, Gardner M-AG, Parizeau M, Gagné C. DEAP: Evolutionary algorithms made easy. J Mach Learn Res 2012;13(1):2171–5.

[22] Palm RB. EvoStrat. 2020, https://github.com/rasmusbergpalm/evostrat.

[23] Nomura M, Shibata M. Cmaes: A simple yet practical Python library for CMA-ES. 2024, arXiv preprint arXiv:2402.01373.

[24] Gad AF. Pygad: An intuitive genetic algorithm python library. Multimedia Tools Appl 2023;1–14.

[25] Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol J-D, Muller EB, et al. BluePyOpt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. Front Neuroinform 2016;10(17). http://dx.doi.org/10.3389/fninf.2016.00017, URL http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2016.00017/abstract.

[26] Sipper M, Halperin T, Tzruia I, Elyasaf A. EC-kity: Evolutionary computation tool kit in Python with seamless machine learning integration. SoftwareX 2023;22:101381.

[27] Qiskit. UMDA. https://docs.quantum.ibm.com/api/qiskit/qiskit.algorithms.optimizers.UMDA.

[28] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.
[29] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[30] Nielsen MA, Chuang IL. Quantum computation and quantum information, vol. 2, Cambridge university press Cambridge; 2001.
[31] Benedetti M, Lloyd E, Sack S, Fiorentini M. Parameterized quantum circuits as machine learning models. Quantum Sci Technol 2019;4(4):043001.
[32] Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm. 2014, arXiv preprint arXiv:1411.4028.