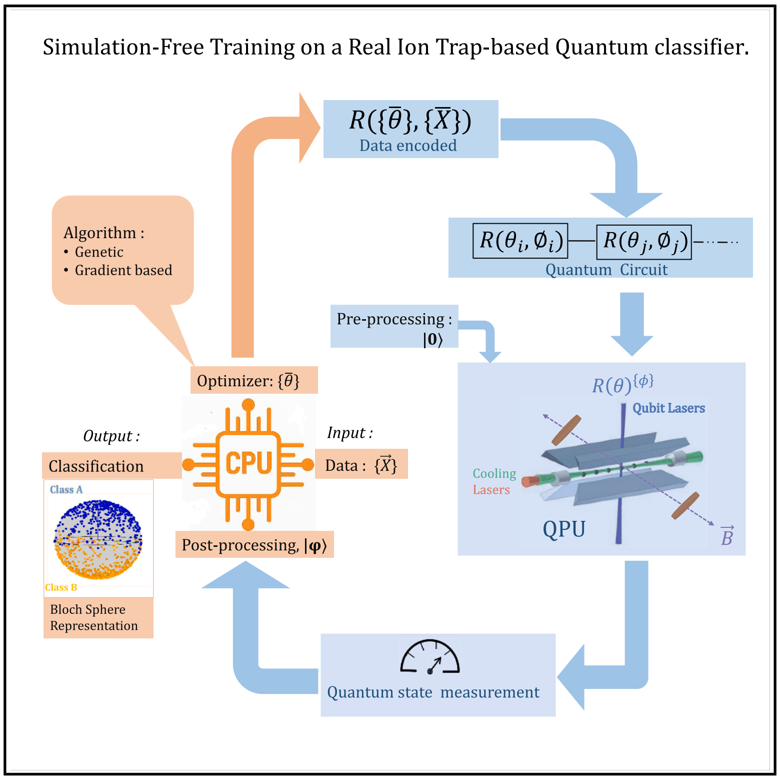
iScience

Article

Practicality of training a quantum-classical machine in the noisy intermediate-scale quantum era

Graphical abstract


Simulation-Free Training on a Real Ion Trap-based Quantum classifier.
$R(\{\bar{\theta}\},\{\bar{X}\})$
Data encoded
$R(\theta_i,\phi_i)$ — $R(\theta_j,\phi_j)$
Quantum Circuit
Algorithm :
• Genetic
• Gradient based
Pre-processing : $|0\rangle$
Optimizer: $\{\bar{\theta}\}$
$R(\theta)^{\{\phi\}}$
Qubit Lasers
Output :
Classification
Input :
Data : $\{\vec{X}\}$
CPU
Cooling Lasers
Class A
Class B
Bloch Sphere Representation
Post-processing, $|\varphi\rangle$
QPU
$\vec{B}$
Quantum state measurement

Highlights

- Simulation-free training conducted on a real ion-trap quantum system
- Genetic algorithms outperform gradient methods in optimization on NISQ hardware
- Demonstrated reliability of hybrid training for binary classification task
- Hardware-aware strategies developed for practical QML applications

Authors

Tarun Dutta, Alex Jin, Clarence Liu Huihong, José Ignacio Latorre, Manas Mukherjee

Correspondence

tarunduttaz@gmail.com (T.D.), alex_jin@u.nus.edu (A.J.), manas.mukh@gmail.com (M.M.)

In brief

Physics; Computer science

CellPress
OPEN ACCESS

Article

Practicality of training a quantum-classical machine in the noisy intermediate-scale quantum era

Tarun Dutta,[1,5,6,*] Alex Jin,[1,*] Clarence Liu Huihong,[1] José Ignacio Latorre,[1,2,3] and Manas Mukherjee[1,4,*]
[1]Centre for Quantum Technologies, National University of Singapore, Singapore 117543, Singapore
[2]Qilimanjaro Quantum Tech, 08007 Barcelona, Spain
[3]Quantum Research Centre, Technology Innovation Institute, P.O.Box: 9639, Abu Dhabi, United Arab Emirates
[4]Institute of Material Research and Engineering (IMRE), Agency for Science, Technology and Research (A*STAR), 2 Fusionopolis Way, Innovis 08-03, Singapore 138634, Republic of Singapore
[5]Present address: School of Physics, University of Hyderabad, Hyderabad 500046, India
[6]Lead contact
*Correspondence: tarunduttaz@gmail.com (T.D.), alex_jin@u.nus.edu (A.J.), manas.mukh@gmail.com (M.M.)
https://doi.org/10.1016/j.isci.2025.113058

SUMMARY

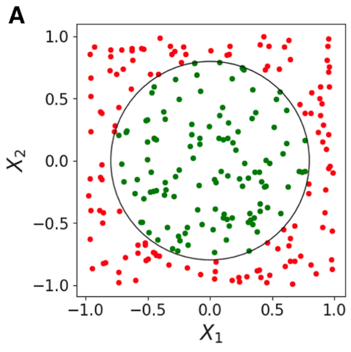
Advancements in classical computing have propelled machine learning applications, yet inherent limitations persist in terms of energy, resource, and speed. Quantum machine learning offers a promising avenue to overcome these limitations but poses its own hurdles. This experimental study investigates the training limits of a real quantum-classical hybrid system on an ion-trap platform using supervised protocols. It addresses the challenges of coupling ion-trap systems with classical processors and highlights the effectiveness of genetic algorithms for NISQ devices and complex binary classification tasks with many local minima. The analysis reveals why gradient-based methods may not be ideal in the NISQ era. Our results provide insights into optimizing hybrid quantum-classical systems, emphasizing the importance of training strategies and hardware design. This work not only enhances understanding of such systems but also illustrates their practical potential in solving real-world problems without relying on classical simulators, bridging the gap between quantum and classical computing paradigms.

INTRODUCTION

The rapid progress in classical computing hardware has significantly advanced the field of machine learning, addressing a diverse range of real-world problems.[1–5] However, inherent limitations in speed and energy efficiency persist, as dictated by Moore's and Koomey's laws,[6–8] respectively. Quantum machine learning (QML) algorithms hold promise in overcoming these barriers, particularly as quantum hardware evolves to handle complex and practically useful problems.[9,10] As quantum hardware scales up, understanding system-level implementation becomes paramount, especially in small-scale systems without classical simulation assistance. This pursuit unravels unique challenges in quantum and hybrid systems, crucial given quantum computing's inherent parallelism and nonlinear dynamics.

Quantum computers, although limited in computational capacity, are steadily advancing in the Noisy Intermediate-Scale Quantum (NISQ) era.[11–16] They excel in certain areas, particularly quantum-classical hybrid algorithms,[17] which synergetically integrate quantum and classical subroutines in the most efficient manner, as is done in the variational quantum algorithm (VQA). In QML, quantum neural networks (QNN) compute cost functions using quantum hardware and optimize subroutines on classical computers.[18] Empirical comparisons between QNN and classical neural networks (CNNs) are essential due to the absence of analytical models for heuristic-based algorithms. Recognizing limitations, such as communication latency, quantum hardware precision, choice of optimizer, training duration, and resource allocation, are crucial for these hybrid systems in the NISQ era.

In most of the current implementations of QML algorithms, the training of the quantum machine has been performed either in a classical simulator[19,20] or with significant guidance from a classical machine.[21] In the latter case, the quantum machine is trained only in the vicinity of an optimal solution. Currently, the small system size allows efficient classical simulation to guide the quantum machine in training, but eventually a large-scale quantum-classical hybrid system needs to be trained by itself without any classical simulation guidance. Unlike classical machine learning, classical simulation guided supervised learning is unique to quantum systems. The challenges in implementing unguided training on a quantum machine are multi-fold: (1) transfer of classical data to a quantum computer, (2) drifts in operational parameters of a quantum computer, (3) the selection of an efficient classical optimizer, and (4) latency inherent in the classical-quantum interface. In this manuscript, we present the implementation and comprehensive analysis of a training



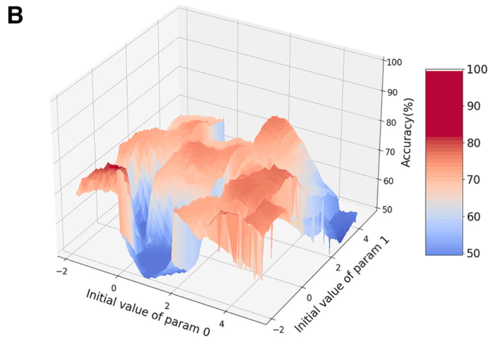


Figure 1. Binary classification problem and its landscape

(A) Binary classification problem statement: the task for the machine is to assign a label (red/green) to any arbitrary point on a two-dimensional space based on its coordinate whether inside or outside the pre-defined circle (black line) respectively.

(B) Problem landscape: highest achievable accuracy in the neighborhood of a randomly selected parameter vector using Ansatz-2C, details of ansatz described in the section (choice of kernel). The optimal accuracy of the neighborhood is plotted against the first two parameter values. The landscape is filled with local minima and maxima, and tiny perturbation in one parameter could lead to sharp drop of the classifier's accuracy.

protocol for a quantum classifier, leveraging a hybrid architecture that combines an ion trap-based quantum system with a classical processor. Our implementation not only addresses the aforementioned challenges but also unveils additional considerations necessary for optimizing the efficiency and enhancing the performance of the hybrid system beyond that of classical solutions.

The choice of an appropriate classical optimizer is often associated with the specific nature of the problem and the topology of the hyperparameter space. While this correlation is well-established, as evidenced by the phenomenon of barren plateau,[22] in the case of quantum systems in the NISQ era the challenges are compounded by the presence of noise and systematic biases. We show that a genetic algorithm (GA), based on the concept of biological evolution, outperforms various gradient-based approaches when deployed on actual quantum hardware systems that are noisy. The GA considers the quantum hardware as a black-box with inherent noise and bias, but as long as the drift of the operational point remains low, the training process converges rapidly. On the contrary, the measurement postulate of quantum mechanics forbids gradient measurement without affecting the intermediate states of the quantum computer. Consequently, gradient-based optimization strategies incur significant temporal and resource costs. Uploading classical data to a quantum computer poses a major challenge, especially in machine learning applications where training dominates computational time. Previous quantum machine learning implementations typically conduct training on a classical simulator, reserving the QML system for validation.[23] This is feasible due to the small size of NISQ quantum hardware, allowing classical simulation. Classical training data are usually mapped to quantum states' amplitude or phase,[24] or it can in-principle be uploaded to a quantum random access memory.[25,26] While the former approach has been implemented on various platforms with specific circuit depth requirements,[24] the latter is yet to be experimentally realized due to technical complexity. In this study, we employ a third approach, directly uploading classical data into quantum gate parameters using the data re-uploading algorithm.[27] Previous work validated QNN using this algorithm,[23] although optimal machine learnable parameters (MLP) were derived from training on a classical simulator due to the cost and accuracy limitations of training on NISQ machines.

In the following, we introduce a binary classification problem and show the result of training conducted on a hybrid quantum-classical (HQC) system based on an ion-trap platform without the guidance of any classical simulation. The final accuracy and the rate of convergence of the three training routines are compared in a real experimental set-up. We argue that the GA outperforms gradient-based optimizers for this type of problem where the hyperparameter space is replete with many local minima as seen in Figure 1. It is also important to recognize that the performance of the classical subroutine depends on the choice of ansatz used for the quantum subroutine. To this end, we have conducted a comprehensive analysis of the hybrid system to ensure that the training process is both efficient and accurate. The section *method details* in the STAR Methods section provides extensive details on the experimental procedures, highlighting key aspects of the classical and quantum hardware that are essential for the effective training of the hybrid system.

RESULTS

A classifier is an abstract map, $f:\mathbb{R}^n \rightarrow \mathbb{C}$, from the Euclidean space (where data resides) $\mathbb{R}^n$ to a finite set $\mathbb{C}$, which represents the individual classes we are trying to classify. The goal is to train the map f by varying certain variables (also called machine learnable parameters) of the map, such that data of a certain type or label maps to region in $\mathbb{C}$ space which is clearly separable from data with all other labels.

In the data re-uploading quantum classifier paradigm, we define a sequence of unitary operations U_l applied to an initial state $|0\rangle$, such that the final state $|\phi\rangle$ is:

$$|\phi\rangle = U_L(\overline{\theta},\overline{x})U_{L-1}(\overline{\theta},\overline{x})\ldots U_0(\overline{\theta},\overline{x})|0\rangle = \prod_{l=0}^{L} U_l(\overline{\theta},\overline{x})|0\rangle \qquad \text{(Equation 1)}$$

Our quantum circuit processes a vector $\overline{x} \in \mathbb{R}^n$, combined with variational parameters $\overline{\theta}$, creating a sequence of unitary operators applied to an initial state $|0\rangle$. The output state is measured and classified as A (green squares in Figure 1) or B (red dots in Figure 1) based on predefined criteria in the Hilbert space. The classifier, utilizing the data re-uploading algorithm, has been benchmarked against CNNs,[23] showing promise

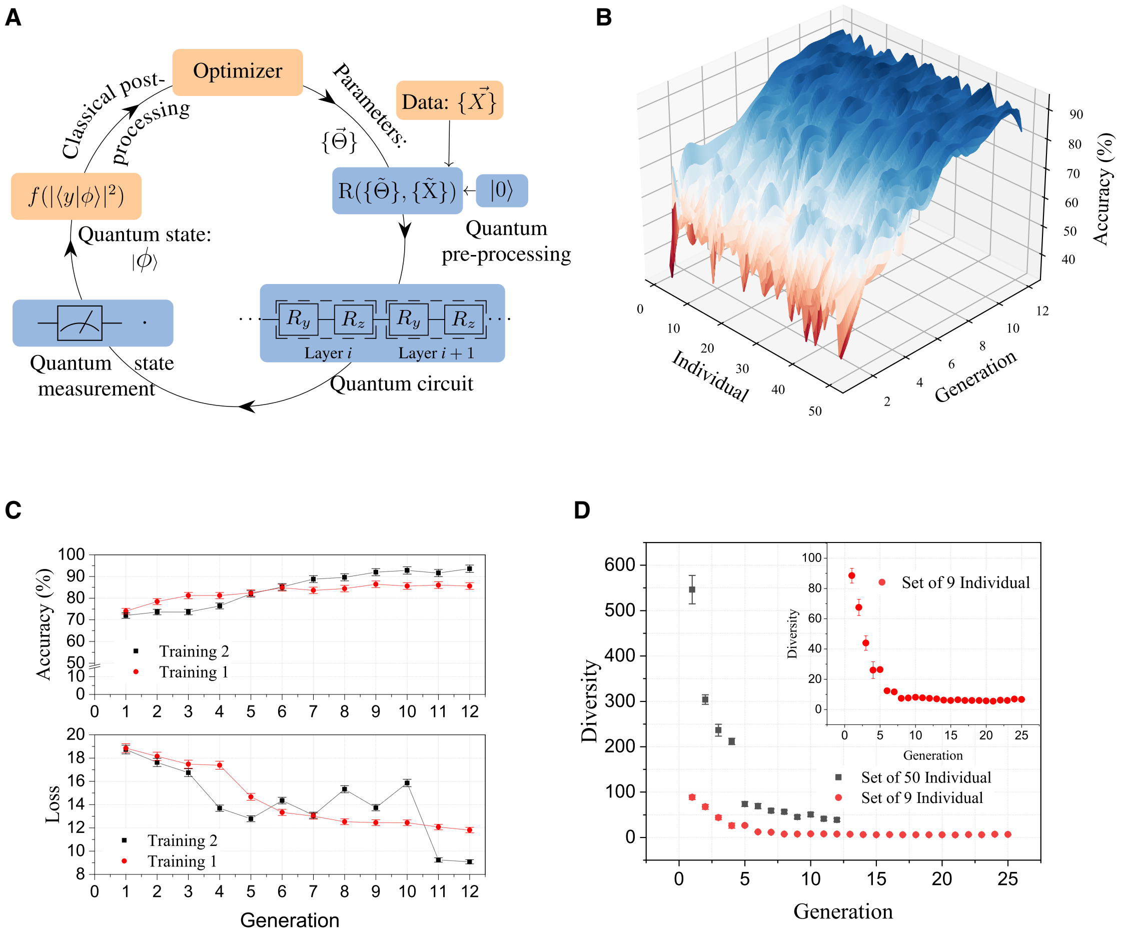


Figure 2. Quantum-classical training protocol and performance analysis

(A) This concise circular representation highlights the quantum training process for classification using classical and quantum components (boxes filled in orange and blue color respectively). The flow illustrates the path of data and operations, starting with classical pre-processing, moving to quantum processing, and concluding with classical post-processing and model updating. Arrows indicate the direction of data flow and processing steps. See the text for details.

(B) Search for optimal parameters for classification task using GA: The ion trap-based QPU is used for training on 250 random data points. The depth of the circuit is kept fixed to 4 layers as it is sufficient to classify the current problem with accuracy above 90%. For this training, a set of 50 individuals have been used and the data is uploaded using Ansatz 2C. A detailed discussion is in the STAR Methods section.

(C) The best accuracies and losses (cross entropy) for each generation from two independent experimental runs, labeled as Training 1 and Training 2, are displayed here. These experiments were performed using the same QPU. The best accuracy achieved is 93.6 ± 2%.

(D) The diversity of individuals (measured as the sum of pairwise L_2-*norm*) varies from one generation to the next. When starting with 50 individuals, the initial diversity is significantly higher compared to beginning with only 9 individuals. The inset provides a magnified view of the relationship between diversity and generation for a set of 9 individuals. A higher initial diversity is associated with achieving higher accuracy with fewer generations.

despite limited parameter space availability. For training, we selected the circle as a test-bed boundary, offering a dense array of local minima, ideal for testing as depicted in Figure 1.

A set of 250 random data (points in a 2D plane) is chosen for the training procedure. As illustrated in Figure 2A, training involves a repetitive sequence of the following steps: (1) classical optimizer generating a set of variational parameters, (2) mapping of data and variational parameters to gate parameters, (3) sequential uploading of mapped parameters to the ion trap quantum computer, (4) projection measurement on the quantum computer and cost function evaluation on the classical computer, (5) optimizing on classical computer and finally closing the loop by returning to step (1). The sequence (1–4) remains consistent throughout this work, with the only variation being in the classical optimizers. This approach allows for a direct comparison of each optimizer under the same conditions. One of the most crucial steps in this process is (2) which differentiates the challenges in an analogue simulator[24] to a circuit-based quantum computer like ours.

Genetic algorithm

The GA is a family of optimization algorithms inspired by the process of natural selection, emphasizing the survival of the fittest principle. According to a survey,[28] there are 4 common degrees of freedom or hyperparameters in a GA. Each of these hyperparameters as applicable to the quantum classifier has been investigated in the STAR Methods section (genetic optimizer). Once selected, these parameters remain unchanged throughout classifier training. The key training outcome is depicted in Figure 2C. Using a population size of only 50, a training accuracy exceeding 90% is achieved just within nine generations of the training routine. The figure presents two experimental sample training runs, illustrating the evolutionary process and its robustness. The evolution of the accuracy with generations denoted by the ● points initially improves until the 6th generation, but then remains at 83% even after the 10th generation, while ■ points reach over 90% accuracy following a different path. The uncertainty of experimental data is ±2%. The error represents the standard deviation of 10 repeated trials conducted on the same dataset, indicating that underlying systematic uncertainties contribute to the overall uncertainty in the accuracy, as detailed in.[23] Additionally, the plot illustrates the evolution of cross-entropy loss across generations, indicating iterative refinement and improved accuracy alignment.

The experimental training run (shown in ■ in Figure 2C) is further analyzed in terms of the individuals' (in a population of 50) evolutionary path over generations, as illustrated in Figure 2B. The colormap shows the training accuracy of each individual in the population for a given generation, starting from a wide range of distribution (denoted by the spread of the color). To facilitate an easier interpretation of the colormap, the values between discrete measurement points (at integer values of population number and generation number) are interpolated and smoothed. Note that the GA in each generation creates a new set of population based on the algorithm and hyperparameter values, as discussed earlier.

When the mutation probability is held constant, the optimizer tends to explore only local optima, resulting in a lower final accuracy compared to scenarios where the mutation probability is dynamically adjusted based on the generation number (Figure S8). This phenomenon can be attributed to the inherent nature of GAs, where a diverse population allows for a more global search across the solution landscape. Conversely, maintaining a static mutation rate throughout the learning process impedes the algorithmic convergence, leading to sub-optimal exploration (section genetic optimizer). In the context of the quantum-classical hybrid system, the large exploration space as well as the dynamical adjustment of the hyperparameter helps in dynamically mitigating any bias that arises in the quantum system over time.

Contrasting with the accuracy attained by GA runs with a population of 50, the maximum accuracy achieved by runs with a smaller population of just 9 was approximately 85%, even after the 25th generation. The landscape depicting the evolution of accuracies over generations for this individual is presented in Figure S9.

Having a lower number of individuals as shown in Figure 2D introduces a different set of challenges. In our experiment, we observed that the rate of convergence differs when comparing populations of 9 and 50 individuals per generation during training. With a population of 9, diversity is initially low, which leads to rapid convergence; however, both the accuracy and final diversity are inferior to what is achieved with a population of 50. This suggests that smaller populations may lead to stronger exploitation of the search space at the expense of exploration.

So far, we have been discussing the impact of GA on the rate of convergence of a hybrid quantum-classical training system. However, the most commonly used (classical) optimizer for these systems is based on gradient descent (GD) algorithm. In the following, we show the learning performance of the HQC using a few flavors of GDs.

Gradient-based algorithms

Gradient-based method is an iterative optimization method that leverages on first and possibly higher order derivatives of the cost function with respect to MLPs. Here, we explore the application a quasi-Newton method, specifically the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.[29] We focused our studies to the BFGS algorithm and tested it with two different gradient evaluation methods as discussed in the following sections. GD and stochastic gradient descent (SGD) are well-established techniques in the fields of statistics and classical deep learning and have been previously employed in optimizing random quantum circuits.[30] However, within our NISQ setup, these methods have demonstrated less reliable convergence and hence only discussed in the STAR Methods section (gradient-based optimizer) as a comparison.

The BFGS method is a prominent iterative technique for tackling unconstrained nonlinear optimization problems. It incorporates second-order derivative information to guide the optimization path, resulting in a more reliable convergence trajectory compared to methods that rely solely on first-order gradients. As a member of the quasi-Newton family, BFGS approximates the Hessian matrix, avoiding the computational expense of calculating the Hessian directly. This cost-effectiveness is especially beneficial for training routines on our NISQ systems. In contrast to classical machine learning algorithms, where gradients are typically computed analytically via back-propagation, quantum systems necessitate measuring the quantum processor to obtain gradients with respect to specific parameters. In the following sections, we discuss the implementation of two distinct methods of gradient estimation on a QPU within a 14-dimensional parameter space designed for machine learning. We then conducted a comparative analysis of the performance of these two gradient estimation techniques.

Methods of gradient estimation

The BFGS method is renowned for its superior convergence properties. In order to leverage this algorithm, it remains to estimate the gradient at each desired $\overline{\theta}$ (see section gradient-based optimizer) value. In classical machine learning problems, BFGS-based methods usually approximate the derivatives using the finite difference approach.

Generally, any gradient-based algorithms will perform better when the step size of finite difference computation is small. In our problem setting, we found through simulation that a step size below 0.05 is needed. However, achieving such a small step size is challenging in practice with the NISQ devices. To address this issue, we employ a different technique known as the *parameter shift method*.[31] This technique evaluates gradient of any parameter using arbitrary shifts in two opposite directions. We fix shift sizes of $\pm\frac{\pi}{2}$ in order to maximally

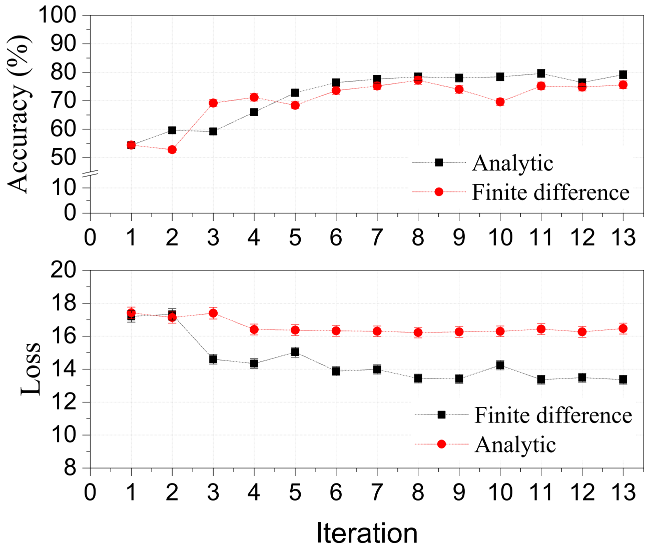


Figure 3. Training accuracy and loss curves for optimization using BFGS

The training accuracy (top) and cross-entropy loss (bottom) on the QPU are graphed against iteration steps, utilizing the BFGS optimizer for both finite difference and analytic approaches. The search for the best parameters for the classification task employed the BFGS-finite difference method with a step size of 0.5. Connected lines between data points are utilized in tracking the data's progression only. Here, we have presented data up to 13 iterations for both the finite difference and analytic methods to facilitate comparison. In reality, we have collected data up to approximately 30 iterations (600 Measurements) to corroborate the measurement with GA, as shown in Figure 4A.

leverage this technique and reduce the effect of random noise. An alternative method, particularly applicable in our context of data re-uploading algorithm is *analytic gradient*. It is possible to find the analytic gradient of our circuit at each point in the parameter space, thus using the QPU, the analytic gradient can be measured without any approximation and iterated successively.

Figure 3 illustrates the training accuracy and cross-entropy loss of our binary classifier, comparing the finite-difference and analytic gradient methods, distinct from the GA. The finite difference gradient method yields lower loss values compared to the analytic gradient approach. By estimating gradients through differences of measurements at paired points, it partially cancels correlated noise and bias. In contrast, the analytic gradient method, relying on single measurements per term, lacks this noise-cancellation effect, making it more sensitive to hardware noise. This underscores the influence of noise characteristics on optimization performance and the need to consider hardware-specific noise when designing optimization methods for QML applications. Notably, after the 6th iteration, both gradient based methods exhibit stagnant accuracy improvements, mirrored in the cross-entropy loss, where its slope remains consistent over subsequent iterations. In contrast, the GA demonstrates a pronounced decline in slope across generations as in Figure 2C. We compare CPU-based simulation results with QPU-based experimental outcomes as shown in Figure 4A, applying the same operational conditions to finite-difference, analytic gradient measurement techniques and GA. Notably, the simulations represent an idealized scenario devoid of noise or bias, which is not the case for the experimental runs. Consequently, the observed discrepancies in the learning curves between simulation and experimental results can be attributed to both bias and noise inherent in the quantum system. In our earlier work,[23] we demonstrated that employing a classically bootstrapped learning method on the QPU can enhance the performance.

Overall, the discrepancies observed between the simulation and experimental results cannot be solely attributed to experimental bias. Given that the BFGS method is deterministic, unlike the probabilistic nature of GA, a closer alignment between the datasets is anticipated. However, the experimental data points consistently exhibit lower accuracy compared to the idealized, noise-free simulation. This divergence is likely due to factors such as bias, decoherence, and inherent noise within the quantum subroutine. To mitigate these issues, we have explored post-measurement error correction techniques, which are detailed in the STAR Methods section (algorithmic error in NISQ era). However, the GA is robust, in particular with respect to bias which is evident from the good match between simulated trajectory (blue defused band) and the two experimental runs (●). On the contrary, we observe that the BFGS gradient experiments (●, ●) under-performs as compared to the CPU simulation (orange defused band). One interesting observation in the BFGS analytic gradient experiment is the sudden drop in the accuracy around 200th measurement, which we attribute to the existence of the cliffs in the parameter space as shown and discussed in Figure 1 at which a small bias in the experiment leads to a sharp fall in the gradient, followed by a gradual recovery. In some conditions we have also observed this in the ideal simulations. Finally, as a comparison of ideal simulation with smallest stepsize (0.001) we show in Figure 4B that both GA and BFGS are comparable in terms of achievable accuracy and the time to converge but GD shows slow convergence. This indicates that BFGS methods are only as good as GA if and only if the QPU is noiseless.

Our findings indicate that the final training accuracy is largely unaffected by the choice of gradient evaluation method, whether analytical or finite-difference-based. In both instances, the final accuracy falls short of that achieved by GAs. It is also worth noting that the data re-uploading algorithm, which employs rotational gates dependent on sine and cosine functions, allows for analytic gradient evaluation that is less resource-intensive compared to other variational quantum algorithms.

After implementing both GA and gradient-based optimizers for HQC in the NISQ era, a cross-algorithm performance comparison is warranted. Key performance metrics for classifier training should include the rate of convergence, the final achievable accuracy, and the associated resource requirements. We will discuss each of these performance indicators in the following section.

DISCUSSION

In classical machine learning frameworks, the convergence rate of the MLPs is characterized by the rate at which the algorithm

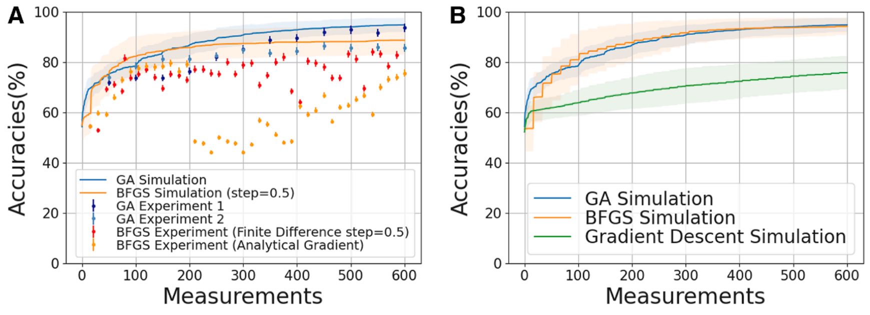


Figure 4. Comparison of optimization techniques

(A) Comparison between simulation and experimental results for genetic algorithm and BFGS: The BFGS simulation run was modified to use a larger step size of 0.5 in the finite difference gradient approximation to match our experimental setup. BFGS-based experimental runs suffer from precision issues due to system noise as explained in the sec. (gradient-based optimizer). Referring to the problem landscape illustrated in Figure 1, we can see that a small perturbation from noise could significantly alter the direction and magnitude of our gradient. GA-based runs are more insensitive to noise and converge similarly as in the case of simulation.

(B) Simulation comparison between genetic and gradient-based optimizers: Gradient descent using first order derivative seems to converge less rapidly in this setup. L-BFGS-B variant was used and it gives comparable performance to that of genetic algorithm. Plots for simulation consist of the average (solid lines) and one standard deviation (semi-transparent band) over multiple runs. The plots for the experiment consist of individual runs (solid dots) and experimental uncertainties (error bars).

refines its parameter estimates, as determined by the cost function. In contrast, the comparison of optimization methods such as GA and gradient-based optimizers in HQCs necessitates a precise definition of a "computational cycle". For the purposes of this study, we define a computational cycle in the context of HQC as the rate of model improvement per projection measurement, where each measurement consists of 100 experimental realizations, commonly referred to as "shots".

Empirical results from our comparative analysis clearly demonstrate that GA optimizers surpass gradient-based optimizers in terms of both convergence rate and final accuracy under our problem setup using NISQ devices. To further illustrate these differences, we include additional numerical experiments. In the Figure S14, we compare the performance of a typical GA solver with that of the L-BFGS-B solver, evaluated under various step sizes for finite-difference gradient estimation. While L-BFGS-B solvers perform well in noise-free environments with smaller step sizes (Figures S14B and S14C), their performance drastically degrades with the introduction of noise. This is expected as the absolute magnitudes of the gradients have magnitudes similar to those of the noise level.

Figure 5 further highlights the superiority of GA over gradient-based optimization by comparing the performance of the GA solver and the L-BFGS-B solver under Gaussian noise of 0.05, consistent with our experimental setup. We also include noise-free results alongside the noisy simulations, revealing a visibly larger performance gap for the L-BFGS-B solver in the presence of noise.

A critical factor influencing the performance limitations of gradient-based methods is the resolution of the step size. In the context of the binary classifier, the MLP parameter space is replete with local minima, which poses a significant challenge for gradient-based optimization. In contrast, GAs leverage a population-based approach to explore multiple regions of the MLP space concurrently. This diversity allows for a more comprehensive search and exploitation of multiple promising regions. Consequently, GAs exhibit a degree of robustness against the noise and systematic biases that may be present in the quantum subroutine, which are factors that can detrimentally affect the performance of gradient-based methods. The phenomenon of *Barren Plateaus* presents a significant challenge in the field of variational quantum computing, where the optimization landscape becomes exponentially flat, hindering the training of quantum circuits. This issue is often mitigated by narrowing the search space of the problem.[32] We wish to highlight that the intrinsic capability of GA to concurrently explore multiple regions of the search space renders them less susceptible to the Barren Plateaus problem.[33]

We now focus on the error mitigation strategy utilized in our implementation. As depicted in Figure S13, a correlation plot contrasts the simulated data with the experimental outputs of the cost function across each training iteration. Under ideal conditions, this relationship would be characterized by a linear correlation with a slope of 45° (represented by a dashed line). However, our observations indicate a dispersion of data points around this idealized line, accompanied by an apparent offset and altered slope. A histogram of the deviations from the ideal linear relationship reveals that the experimental error is predominantly Gaussian noise. Further analysis suggests that the altered slope of the correlation curve may be attributable to qubit decoherence, which introduces an undesirable systematic bias. To rectify this bias, we propose the utilization of the inverse correlation matrix as a compensatory measure applied to the experimental data. Despite this corrective approach, we note that it does not influence the gradient estimations in the context of binary classification problems, which follows from the mathematical definition of our cost functions as listed in Equations 6, 7, and 8 in the section (*cost function*). A thorough study of this error mitigation technique and its implications is provided in the section (mitigation of random error). Furthermore, we would like to point out that the ideal step size required to improve on the gradient-based approaches can be derived from the algorithmic error, setting the limits on the resolution of the parameter step size that can be practically applied to find the gradient as illustrated in Figure S16.

In an HQC system, establishing quantum advantage fundamentally hinges on demonstrating superior speed or energy efficiency. However, quantifying these attributes presents several challenges: (1) The heuristic nature of neural network-based

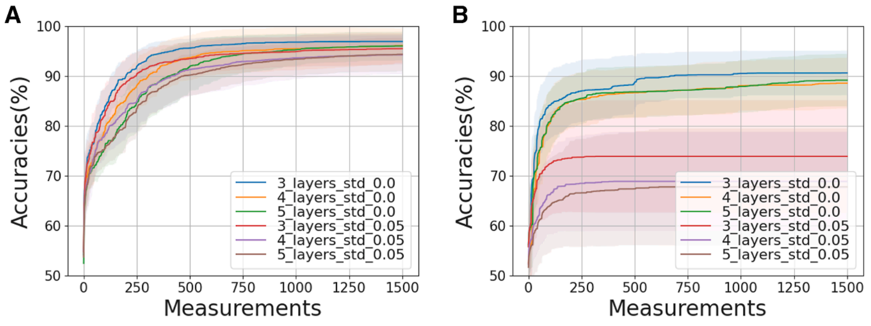


Figure 5. Comparative study of a binary classification problem using noiseless and noisy ($\sim \mathcal{N}(0, 0.05)$), matching our experimental setup) quantum computer

(A) Genetic algorithm solver with population size of 50, scattered crossover, exponentially decaying mutation rate, and SSS scheme. Details of the population size, mutation rate, and SSS scheme are provided in Sec. (genetic optimizer).

(B) L-BFGS-B solver with a step size of 0.5 (for gradient estimation) used for the experiment. The experimental results are shown in Figure 4A for comparison.

machine learning models precludes a straightforward assessment of computational resources from complexity theory. (2) In a hybrid system, the total computation time must account for the communication latency between the quantum and classical subroutines, which can significantly impact overall performance. (3) The assessment of energy efficiency must encompass both quantum and classical subroutines. However, the lack of a standardized framework for comparing these disparate computational architectures complicates this task, as they utilize fundamentally different physical resources. Despite these challenges, our research aims to demonstrate the capabilities of an ion trap-based HQC system in conducting supervised training with a data re-uploading paradigm, achieving classification accuracy exceeding 90%. Presently, the computational time within our system is predominantly constrained by the communication overhead between the quantum and classical subroutines. Potential improvements could be realized by integrating more substantial classical memory and computational resources proximal to the quantum hardware controls. Nonetheless, such enhancements may not be universally applicable, particularly in quantum computing architectures that operate at cryogenic temperatures.

In summary, we conclude that an ion trap-based hybrid quantum-classical NISQ processor is trainable by successfully executing supervised learning tasks for classification, without guided by classical simulation, achieving final accuracy exceeding 90%. Moreover, through comparative analysis, we have identified the GA as the superior classical optimizer subroutine for the data re-uploading classifier in the context of current NISQ processors, attributed to the limited precision in parameter settings inherent to NISQ technology. Lastly, we emphasize the significance to consider communication overhead between quantum and classical components in the overall performance evaluation, as it presents a potential bottleneck in hybrid NISQ processors.

Limitations of the study

This study employs a minimal setup, a single-qubit classifier and a simple binary classification task, which limits the direct generalizability of the results to larger and more complex quantum systems. Additionally, the effectiveness of optimization strategies like GAs is known to be problem-specific, and further evidence is needed to confirm their scalability and robustness in broader contexts. Nevertheless, this controlled framework provides a clear proof-of-principle for direct hardware training and offers foundational insights relevant for scaling up QML in the NISQ era.

RESOURCE AVAILABILITY

Lead contact

Requests for further information and resources should be directed to and will be fulfilled by the lead contact, Tarun Dutta (tarunduttaz@gmail.com).

Materials availability

This study did not generate new materials.

Data and code availability

- All data reported in this paper may be shared by the corresponding author upon reasonable request.
- This paper does not report original code.
- Any additional information, including code required to reanalyze the data reported in this paper, is available from the lead contact upon reasonable request.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under its Quantum Engineering Program ((NRF2021-QEP2-02-P10).

AUTHOR CONTRIBUTIONS

M.M. and T.D. conceptualized the experiment. T.D. designed and performed the experiment. A.J. led the simulation and computational work. T.D. and A. J. conducted the data analysis and C.L.H. contributed to the recording of experimental data. T.D., A.J., and M.M. wrote the manuscript. M.M. and J.I. L. supervised the work.

DECLARATION OF INTERESTS

The authors declare no competing interests.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- METHOD DETAILS
  - Quantum classifier training methodology
  - Experimental set-up
  - Error sources
  - Time budget
- QUANTIFICATION AND STATISTICAL ANALYSIS
  - Efficient training of NISQ classifier
  - Optimization strategies
  - Algorithmic error in NISQ era
  - Parameter-shift rule and limitation of gradient-based algorithms

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.isci.2025.113058.

Received: March 13, 2025
Revised: May 15, 2025
Accepted: July 2, 2025
Published: July 9, 2025

REFERENCES

1. Libbrecht, M.W., and Noble, W.S. (2015). Machine learning applications in genetics and genomics. Nat. Rev. Genet. *16*, 321–332. https://doi.org/10.1038/nrg3920.
2. Vamathevan, J., Clark, D., Czodrowski, P., Dunham, I., Ferran, E., Lee, G., Li, B., Madabhushi, A., Shah, P., Spitzer, M., and Zhao, S. (2019). Applications of machine learning in drug discovery and development. Nat. Rev. Drug Discov. *18*, 463–477. https://doi.org/10.1038/s41573-019-0024-5.
3. Thirunavukarasu, A.J., Ting, D.S.J., Elangovan, K., Gutierrez, L., Tan, T.F., and Ting, D.S.W. (2023). Large language models in medicine. Nat. Med. *29*, 1930–1940. https://doi.org/10.1038/s41591-023-02448-8.
4. Thiyagalingam, J., Shankar, M., Fox, G., and Hey, T. (2022). Scientific machine learning benchmarks. Nat. Rev. Phys. *4*, 413–420. https://doi.org/10.1038/s42254-022-00441-7.
5. Xiao, T., Zhai, X., Huang, J., Fan, J., and Zeng, G. (2024). Quantum deep generative prior with programmable quantum circuits. Commun. Phys. *7*, 276. https://doi.org/10.1038/s42005-024-01765-9.
6. Moore, G.E. (1998). Cramming more components onto integrated circuits. Proc. IEEE *86*, 82–85. https://doi.org/10.1109/JPROC.1998.658762.
7. Koomey, J., Berard, S., Sanchez, M., and Wong, H. (2011). Implications of historical trends in the electrical efficiency of computing. IEEE Ann. Hist. Comput. *33*, 46–54. https://doi.org/10.1109/MAHC.2010.28.
8. Moore, G.E. (1995). Lithography and the future of Moore's law. In Proc. SPIE 2439, Integrated Circuit Metrology, Inspection, and Process Control IX, M.H. Bennett, ed. (SPIE), pp. 2–17. https://doi.org/10.1117/12.209195.
9. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. Nature *549*, 195–202. https://doi.org/10.1038/nature23474.
10. Xiao, T., Zhai, X., Wu, X., Fan, J., and Zeng, G. (2023). Practical advantage of quantum machine learning in ghost imaging. Commun. Phys. *6*, 171. https://doi.org/10.1038/s42005-023-01290-1.
11. Kim, Y., Eddins, A., Anand, S., Wei, K.X., van den Berg, E., Rosenblatt, S., Nayfeh, H., Wu, Y., Zaletel, M., Temme, K., and Kandala, A. (2023). Evidence for the utility of quantum computing before fault tolerance. Nature *618*, 500–505. https://doi.org/10.1038/s41586-023-06096-3.
12. Robbiati, M., Sopena, A., Papaluca, A., and Carrazza, S. (2023). Real-time error mitigation for variational optimization on quantum hardware. Preprint at arXiv. https://doi.org/10.48550/arXiv.2311.05680.
13. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., et al. (2019). Quantum supremacy using a programmable superconducting processor. Nature *574*, 505–510.
14. Joshi, M.K., Kokail, C., van Bijnen, R., Kranzl, F., Zache, T.V., Blatt, R., Roos, C.F., and Zoller, P. (2023). Exploring large-scale entanglement in quantum simulation. Nature *624*, 539–544.
15. Moses, S.A., Baldwin, C.H., Allman, M.S., Ancona, R., Ascarrunz, L., Barnes, C., Bartolotta, J., Bjork, B., Blanchard, P., Bohn, M., et al. (2023). A race-track trapped-ion quantum processor. Phys. Rev. X *13*, 041052. https://doi.org/10.1103/PhysRevX.13.041052.
16. Barron, S.V., Egger, D.J., Pelofske, E., Bärtschi, A., Eidenbenz, S., Lehmkuehler, M., and Woerner, S. (2024). Provable bounds for noise-free expectation values computed from noisy samples. Nat. Comput. Sci. *4*, 865–875. https://doi.org/10.1038/s43588-024-00709-1.
17. Bharti, K., Cervera-Lierta, A., Kyaw, T.H., Haug, T., Alperin-Lea, S., Anand, A., Degroote, M., Heimonen, H., Kottmann, J.S., Menke, T., et al. (2022). Noisy intermediate-scale quantum algorithms. Rev. Mod. Phys. *94*, 015004. https://doi.org/10.1103/RevModPhys.94.015004.
18. Beer, K., Bondarenko, D., Farrelly, T., Osborne, T.J., Salzmann, R., Scheiermann, D., and Wolf, R. (2020). Training deep quantum neural networks. Nat. Commun. *11*, 808. https://doi.org/10.1038/s41467-020-14454-2.
19. Rebentrost, P., Mohseni, M., and Lloyd, S. (2014). Quantum support vector machine for big data classification. Phys. Rev. Lett. *113*, 130503. https://doi.org/10.1103/PhysRevLett.113.130503.
20. Cai, X.D., Wu, D., Su, Z.E., Chen, M.C., Wang, X.L., Li, L., Liu, N.L., Lu, C. Y., and Pan, J.W. (2015). Entanglement-based machine learning on a quantum computer. Phys. Rev. Lett. *114*, 110504. https://doi.org/10.1103/PhysRevLett.114.110504.
21. Havlíček, V., Córcoles, A.D., Temme, K., Harrow, A.W., Kandala, A., Chow, J.M., and Gambetta, J.M. (2019). Supervised learning with quantum-enhanced feature spaces. Nature *567*, 209–212. https://doi.org/10.1038/s41586-019-0980-2.
22. McClean, J.R., Boixo, S., Smelyanskiy, V.N., Babbush, R., and Neven, H. (2018). Barren plateaus in quantum neural network training landscapes. Nat. Commun. *9*, 4812. https://doi.org/10.1038/s41467-018-07090-4.
23. Dutta, T., Pérez-Salinas, A., Cheng, J.P.S., Latorre, J.I., and Mukherjee, M. (2022). Single-qubit universal classifier implemented on an ion-trap quantum device. Phys. Rev. A *106*, 012411. https://doi.org/10.1103/PhysRevA.106.012411.
24. Pagano, G., Bapat, A., Becker, P., Collins, K.S., De, A., Hess, P.W., Kaplan, H.B., Kyprianidis, A., Tan, W.L., Baldwin, C., et al. (2020). Quantum approximate optimization of the long-range ising model with a trapped-ion quantum simulator. Proc. Natl. Acad. Sci. USA *117*, 25396–25401. https://doi.org/10.1073/pnas.2006373117.
25. Park, D.K., Petruccione, F., and Rhee, J.K.K. (2019). Circuit-based quantum random access memory for classical data. Sci. Rep. *9*, 3949. https://doi.org/10.1038/s41598-019-40439-3.
26. Giovannetti, V., Lloyd, S., and Maccone, L. (2008). Quantum random access memory. Phys. Rev. Lett. *100*, 160501. https://doi.org/10.1103/PhysRevLett.100.160501.
27. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., and Latorre, J.I. (2020). Data re-uploading for a universal quantum classifier. Quantum *4*, 226.
28. Katoch, S., Chauhan, S.S., and Kumar, V. (2021). A review on genetic algorithm: past, present, and future. Multimed. Tools Appl. *80*, 8091–8126.
29. Fletcher, R., and Reeves, C.M. (1964). Function minimization by conjugate gradients. Comput. J. *7*, 149–154. https://doi.org/10.1093/comjnl/7.2.149.
30. Mitarai, K., Negoro, M., Kitagawa, M., and Fujii, K. (2018). Quantum circuit learning. Phys. Rev. A *98*, 032309. https://doi.org/10.1103/PhysRevA.98.032309.
31. Crooks, G.E. (2019). Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. Preprint at arXiv. https://doi.org/10.48550/arXiv.1905.13311.
32. Cerezo, M., Larocca, M., García-Martín, D., Diaz, N., Braccia, P., Fontana, E., Rudolph, M.S., Bermejo, P., Ijaz, A., Thanasilp, S., et al. (2023). Does provable absence of barren plateaus imply classical simulability? or, why we need to rethink variational quantum computing. Preprint at arXiv. https://doi.org/10.48550/arXiv.2312.09121.
33. Acampora, G., Chiatto, A., and Vitiello, A. (2022). Training variational quantum circuits through genetic algorithms. In 2022 IEEE Congress on Evolutionary Computation (CEC) (IEEE), pp. 1–8.
34. Dutta, T., and Mukherjee, M. (2020). A single atom noise probe operating beyond the heisenberg limit. NPJ Quantum Inf. *6*, 3. https://doi.org/10.1038/s41534-019-0234-z.

35. Yum, D., De Munshi, D., Dutta, T., and Mukherjee, M. (2017). Optical barium ion qubit. J. Opt. Soc. Am. B *34*, 1632–1636. http://josab.osa.org/abstract.cfm?URI=josab-34-8-1632.

36. Dutta, T. (2022). An injection-locked 1762 nm laser for trapped barium ion qubits. Appl. Phys. B *128*, 136. https://doi.org/10.1007/s00340-022-07838-3.

37. Ahmadi, M., Dutta, T., and Mukherjee, M. (2024). Scalable narrow linewidth high power laser for barium ion optical qubits. Opt. Express *32*, 17879–17892. https://opg.optica.org/oe/abstract.cfm?URI=oe-32-10-17879.

38. Steinwart, I. (2001). On the influence of the kernel on the consistency of support vector machines. J. Mach. Learn. Res. *2*, 67–93.

39. Smith, L.N. (2018). A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. Preprint at arXiv. https://doi.org/10.48550/arXiv.1803.09820.

40. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. Nature *518*, 529–533.

41. Parameter-shift rules — PennyLane — pennylane.ai. https://pennylane.ai/qml/glossary/parameter_shift/.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and algorithms		
Python version 3.6.0	Python Software Foundation	https://www.python.org

METHOD DETAILS

Here, we provide a detailed description of the quantum classifier based on the data re-uploading algorithm. To facilitate a clear understanding of the performance of a quantum-classical hybrid algorithm in training a quantum classifier, binary classification of points on a plane for a circular boundary is considered here. The approach is based on data re-uploading algorithm[27] which is a quantum equivalent of a classical neural network (CNN). Like any machine learning protocol, the classifier has two modes of operation, namely training and validation. Training is performed on a random set of labeled data to obtain the variational parameters, alternatively called the machine learnable parameters (MLP) of the classifier. Once successfully implemented, these MLPs are used to operate the classifier such that it is able to correctly label any random data into the right class. This mode of operation is called the validation.

Following the *Universal Quantum Circuit Approximation*, we may approximate any classification function up to arbitrary precision by using sufficiently many unitary operations and with the data re-uploaded to each operation. Due to practical reasons, it is more convenient to represent arbitrary unitary operation as a combination of R_y and R_z noncommuting rotational gates of a single qubit. We also note that there are infinitely many ways of combining parameters ($\overline{\theta}$) and data ($\overline{x}$) (the set of all polynomials with 2 variables for example). For practical purposes choosing $\overline{\theta} \in \mathbb{R}^4$ and linearly combining them with $\overline{x}$ worked sufficiently well. It acts as kernel or feature space mapping for the quantum classifier. The quantum classifier circuit used for this comparative study is formed of 4 layers. Thus from Equation 1,

$$|\phi\rangle = U_3(\overline{\theta},\overline{x})U_2(\overline{\theta},\overline{x})U_1(\overline{\theta},\overline{x})U_0(\overline{\theta},\overline{x})|0\rangle \quad \text{(Equation 2)}$$

$$|\phi\rangle = \prod_{l=0}^{4} U_l(\overline{\theta},\overline{x})|0\rangle \quad \text{(Equation 3)}$$

and each U_l consists of an R_y rotation followed by an R_z rotation, whose values are computed according to the particular choice of the kernel. A good choice of the kernel is discussed in sec. (choice of kernel). Given a kernel with L layers and parameters $\overline{\theta}$, a data point $\overline{x}$ and a label y, the measurement function returns the expected value projected onto the label state $|y\rangle$:

$$M(\overline{\theta},\overline{x},y) = \left|\left\langle y \middle| \prod_{l=0}^{L} U_l(\overline{\theta},\overline{x}) \middle| 0 \right\rangle\right|^2 \quad \text{(Equation 4)}$$

In the following, we will briefly discuss the quantum classifier training methodology, the experimental set-up followed by a detailed comparative study of the importance of choice of anzats, optimizers and hyper-parameters.

Quantum classifier training methodology

Similarly to any machine learning protocol, the quantum classifier has two modes of operations namely, training and validation. The supervised training is performed on a random set of labeled data to obtain the variational parameters of the classifier. Once implemented successfully, these optimal parameters are used to operate the classifier such that it is able to correctly label any data into the right class. This mode of operation is called the validation.

Here, we operate the hybrid system in training mode to obtain the optimal set of variational parameters as illustrated in Figure S1. The quantum processing unit is based on ion-trap qubit, encoded in the electronic states of barium ion. In the following, we demonstrate the sequence carried out to optimize a predefined cost function by varying the values of the quantum gate parameters.

Step 1: A set of initial randomize parameters of the circuit are generated in a classical computer. These parameters are convoluted with 250 labeled training data following different possible *Ansatz* (see section choice of kernel) to transform the data to higher dimensional parameter space. This kernel data in the data re-uploading algorithm is uploaded to the quantum computer as gate parameters.
Step 2: The list of gate parameters are loaded to the quantum processor.
Step 3: The state of the quantum processor is initialized to the ground state.
Step 4: The list of gate parameters are sequentially applied as per the circuit.

Step 5: Final state after execution of the circuit is projected on the label state and measurement of the projection is performed.
Step 6: Steps 3–5 are repeated 150 times to obtain the expectation value of the projection for one out of 250 labeled training data.
Step 7: Steps 2–6 are repeated for 250 labeled training data points.
Step 8: A suitable cost function is evaluated from the results in Step 7 which is fed back to the classical optimizer to provide the next set of parameters of the circuit.
Step 9: Once the cost function reaches a pre-defined threshold, the algorithm stops and the final set of parameters is considered as optimal set.

The steps 2–6 are carried out with the trapped ion quantum processor while the rest are performed on a classical computer. Since the algorithm runs in a loop between the quantum and classical processors, the communication time plays an important role. In the following, we will discuss the key steps and their implementation methods.

After the training is complete, a further step has been carried out in the quantum device by using the best optimal parameters obtained from training 2 (using GA optimizer) shown in Figure S2 is the classification of 1000 test data points. The reason to proceed in this way is to check the generalization capabilities of the quantum classifier without facing a too-costly optimization step. At the end of the execution, measurements are made to obtain the relative fidelity between the output state and all label states. The fidelity/accuracy obtained in this case (Figure S2, right) is about 92.8% which is within the error-bar of the training accuracy (Figure S2, left), 93.6 ± 1.8%. The error here refers to one standard deviation of 10 repeated trials performed on the same data-set and reflects the underlying systematic uncertainty leading to an uncertainty of the accuracy. Our experimental results confirm the trend seen by simulation, and the finite coherence and imperfections of the preparation of qubit do not seem to impact results significantly for the shallow circuit considered here.

Next, we introduce the experimental hardware stacks shown in Figure S3, starting with the quantum processor (QPU) and proceeding to the classical processor consisting of the middleware and the CPU.

Experimental set-up

The full-stack quantum classifier hardware can be broken down into three functional stacks, namely the quantum processing unit (QPU), the middleware and the classical processing unit (CPU) as illustrated in Figure S3. The QPU and the middleware is very similar to our earlier work,[23] except that the classical processing unit (CPU) is modified to efficiently perform training of the quantum-classical hybrid classifier for the task of binary classification. The QPU consists of three functional blocks namely, the ion trap and lasers, opto-electronic interface, and the RF drivers. The ions are confined in a linear blade trap with axial confinement frequency $\sim 2\pi \times 0.5$ MHz, and radial mode frequencies $\sim 2\pi \times 1.5$ MHz. A magnetic field of 0.36 mT provided via a low-temperature coefficient $Sm_2\,Co_{17}$ permanent magnets outside the vacuum chamber establishes the quantization direction, which is oriented $\sim 45 deg$ from the trap axis as shown in Figure S3. As in ref.[23,34] we choose $S_{\frac{1}{2},-\frac{1}{2}}$ and $D_{\frac{5}{2},-\frac{1}{2}}$ as the qubit's eigen states. This optical qubit transition frequency is weakly sensitive to magnetic field noise. We apply single qubit gates with a narrow-linewidth 1762 nm laser locked to an ultra-stable cavity leading to a linewidth of $\approx$100 Hz.[35–37] Therefore, the significant contribution to de-phasing comes from magnetic field noise and residual time jitters of the gate pulses. The electro/acousto-optic (EO/AO) layer controls the phase, frequency and amplitude of the laser pulses during a gate implementation. Finally, the hardware control of the EOs and AOs is a combination of stable radio-frequency generators and RF amplifiers which in turn are controlled by the middleware consisting of field programmable gate arrays (FPGA) that produce the sequence of radio-frequency pulses as per the algorithmic instructions. The stable radio-frequency generators are made of direct digital synthesizers (DDS) based on analog device chip AD9958 capable of producing 20–250 MHz with controllable frequency (32 bit resolution), phase (16 bit resolution) and amplitude (10 bit resolution). The FPGA, based on the Altera Cyclone V chip, controls the algorithmic time sequence as well as the measurement of the final state of the qubit.

The quantum processor operates through a sequence comprising initialization, gate implementation, and quantum state determination. Prior to performing each training cycle, the qubit is first Doppler cooled to the Lamb-Dicke regime via a fast dipole transition (between S-P levels shown in the barium energy level diagram in Figure S3 at 493 nm, along with the simultaneous application of a repump laser (between D-P levels) at 650 nm. To cool all three normal modes of motion of the ion in the trap, care has been taken that the Doppler-cooling beam has sufficient overlap with all motional modes. In the following, we outline the specific steps of the QPU that are crucial for training, emphasizing their impact on error and fidelity.

Initialization

After the process of Doppler cooling, the ion population is distributed between the two magnetic sub-states of the ground state, $S_{\frac{1}{2}}(m_j = +\frac{1}{2}$ or $-\frac{1}{2})$, with a slightly higher probability of the ion being in the lower state due to different de-tunings with respect to the cooling light. To initialize the qubit in $m_j = -\frac{1}{2}$ state, the ion is optically pumped by applying resonant 1762 nm light for about 200μs on the $S_{\frac{1}{2},+\frac{1}{2}}$ to $D_{\frac{5}{2},-\frac{1}{2}}$ transition accompanied by a continuous 614 nm light addressing $D_{\frac{5}{2}}$ to $P_{\frac{3}{2}}$. Additionally, 650 nm light is simultaneously applied with the 614 nm light to facilitate the depopulation of the $D_{\frac{3}{2}}$ state since decay from $P_{\frac{3}{2}}$ to $D_{\frac{3}{2}}$ is also allowed.

The state initialization process on the $S_{\frac{1}{2}} - D_{\frac{5}{2}}$ transition is somewhat more complex than using circularly polarized light on the $S_{\frac{1}{2}} - P_{\frac{1}{2}}$ or $P_{\frac{3}{2}}$ transition. However, it offers a distinct advantage. On the $S_{\frac{1}{2}} - D_{\frac{5}{2}}$ transition, the magnetic sub-levels can be distinguished by their distinct frequencies. This makes it possible to achieve high-fidelity initialization without the need to be concerned about geometric alignment with respect to the magnetic field, which could otherwise introduce unwanted polarization components. With this procedure, we get pumping efficiencies exceeding 98.7 ± 0.5% for a single $^{138}Ba^{+}$ ion. Once the qubit is initialized, any single qubit rotational gate is implemented by resonantly driving the qubit with full control over the laser phase, power and laser on time. The key parameters of the trapped-ion system used in this experiment are the qubit coherence time of 5 ms and a Rabi π-time of 12 μs, with a gate fidelity of 98.7%.

Quantum gate implementation

The quantum circuits designed for the training process of the classification task on the quantum processor are depicted at the bottom of the schematic in Figure S1. The quantum circuit is defined as a series of non-commutating rotational gates as elucidated in reference.[23,27] Each layer of circuit consists of two gates, and in our implementation, we utilize a total of 4 layers as it was found to be optimal.[23] Depending on the data to be uploaded, single qubit gates are implemented by resonantly driving the qubit transition between $|0\rangle$ and $|1\rangle$ with well-controlled phase and operation time, while maintaining constant laser frequency and power. The schematic description of the training algorithm employed in this work is illustrated in Figure S1. The optimization process continues until optimal values of $\overline{\Theta}$ are obtained, ensuring a classification accuracy that exceeds 90%. The crucial step after the implementation of the quantum gates as per the algorithm is the quantum state determination which is discussed in the following.

State determination

At the end of each measurement, the final state $|\phi\rangle$'s projection on to the label state $|y\rangle$ is measured, and the result is used to compute the cost function (CE) that quantifies the error made in the classification of the training set. In our experimental setup, the state projection is carried out at the end of each experimental sequence using a Photo-Multiplier Tube (PMT) to register the incoming photon number emitted by the ion within a predefined time window. For a time window of 2 ms, the fluorescence lasers at 493 nm (along with 650 nm) are switched on and all photons detected perpendicular to the laser propagation direction during that time are added up by a digital counter in the middleware. If the number of detected photons is above a predefined threshold, we assign the $S_{\frac{1}{2}}$ state to the result, if it is below we assign the $D_{\frac{5}{2}}$ state. The number of events above/below the threshold relative to the total amount of repetitions determines the probabilities $p1$ and $p0$ (Equation 14) corresponding to population of the two states of the qubit. A typical histogram of one of our measurements, depicting the counts acquired in 2 milliseconds with 200 repetitions, is illustrated in Figure S12. After collecting the fluorescence for an integration time of 2 ms, we use aforesaid threshold to determine the state of the ion, discriminating the quantum state with more than 99% accuracy. Achieving high detection fidelity comes at the cost of long state detection time leading to errors. These errors include finite spontaneous decay probability of the $D_{\frac{5}{2}}$ state during the detection period, dark counts of the photo-detector, scattered photons from the laser beam, and Poisson statistics of the bright state. Since, an ideal classifier should operate in a noisy environment, next, we will analyse the classification error rather than the systematic errors of individual gates.

Error sources

Some errors in quantum computers are coherent, implying that imperfect operations remain Hermitian, and applying their inverse restores the system to its previous state. Examples of such errors encompass instances where qubits experience rotations with inaccurately determined Rabi or resonant frequencies. The impact of these issues is reflected in the Figure S13, yet these errors can be mitigated through the implementation of enhanced control schemes. Improvements can be realized through optimized beam delivery or the application of beam pulse shaping techniques. In sec. (algorithmic error in NISQ era), we provide an in-depth account of our efforts to rectify these errors and detail the shortcomings.

The fidelity of quantum computation faces limitations due to experimental noise, causing the system to deviate from its ideal evolution. This deviation can originate from various sources beyond SPAM errors, which are also examined in detail.

Time budget

Training quantum computers can be a time-consuming process, especially when dealing with a quantum machine that is noisy. We are utilizing a hybrid quantum-classical approach where classical computations are used in conjunction with quantum computations to speed up training and optimization processes, still, the time it takes to train our QPU for each generation is about 330 min, each generation has a set of 50 individuals. So, in our case, the current bottleneck is the time spent on data transfer between QPU and CPU via USB is significantly high compared to other aspects of quantum computing, such as quantum gate implementation, cooling, preparation, and measurement time. The details of the time budget for our setup are presented in Figure S4. This can be mitigated by efficient design of the middleware. The data re-uploading algorithm avoids directly loading classical data to the quantum states thereby saving the circuit depth. However, it needs commensurate classical memory in the middleware to continually feed the classical data. Next, we discuss the methods used to improve the efficiency of the classifier training on a hybrid system.

QUANTIFICATION AND STATISTICAL ANALYSIS

Efficient training of NISQ classifier

In the NISQ era, the QPU is noisy but the classical interface is well-developed.

Choice of kernel

The kernel function plays a crucial role in the performance of a classifier as pointed in[38] for support vector machine (SVM) and classical neural networks (CNN). In the data re-uploading setting, the kernel also plays an important role, both in exploration and exploitation of the parameter space. There are four possible kernels applicable to our classifier by considering only linear mapping; these are:

$$\begin{aligned}
\text{Ansatz} - 2\text{A}: &\quad U_i(\overline{\theta},\overline{x}) = R_z(\theta_3)R_y(\theta_0 x_0 + \theta_1 x_1 + \theta_2)\\
\text{Ansatz} - 2\text{B}: &\quad U_i(\overline{\theta},\overline{x}) = R_z(\theta_2 x_1 + \theta_3)R_y(\theta_0 x_0 + \theta_1)\\
\text{Ansatz} - 2\text{C}: &\quad U_i(\overline{\theta},\overline{x}) = R_z(\theta_2 x_0 + \theta_3 x_1)R_y(\theta_0 x_0 + \theta_1 x_1)\\
\text{Ansatz} - 2\text{D}: &\quad U_i(\overline{\theta},\overline{x}) = R_z(\theta_1 x_0 + \theta_2 x_1 + \theta_3)R_y(\theta_0).
\end{aligned}$$

We note that the number 2 in the above alternative kernel label refer to the dimension of the input size, a convention that was introduced in.[27]

To compare the influence of the four choices of kernel, a set of 250 training data points from the binary classification problem are fixed. In Figure S5, we plot the aggregate R_y and R_z rotational values against each other. These plots illustrate the exploration of a part of the feature space based on the kernel used. As we can see, Ansatz 2A and 2D have the least amount of spread because their degree of freedom for R_z and R_y, respectively, are limited. On the other hand, Ansatz 2B and 2C's spreads are more comparable with 2B having slightly more clustering around certain regions. Given the results of the above analysis, we chose Ansatz 2C considering these analysis above during the training of our quantum classifier.

As a precursor to our training algorithm, we note that the particular choice of Ansatz has less impact for gradient-based optimization methods but affect both the rate of convergence and final convergence value for genetic-based optimization methods.

Optimization strategies

With the kernel function fixed, we are ready to proceed to the last two pieces of our algorithm from *Step 8* of Figure S1: the *cost function* and *classical optimizer*. The particular choice of these pieces both determines the rate of convergence and the converged value. Before we dive in, we will concretely define our measurement function. Given a kernel with L layers, parameters $\overline{\theta}$, a data point $\overline{x}$ and a label y, the measurement function returns the expectation value of the final state projected onto the label state:

$$M(\overline{\theta},\overline{x},y) = \left|\left\langle y\middle|\prod_{l=0}^{L}U_l(\overline{\theta},\overline{x})\middle|0\right\rangle\right|^2 \qquad \text{(Equation 5)}$$

We now examine various choices of cost functions and optimizer methods.

Cost functions

A natural way to define the cost function is the accuracy of our quantum classifier since we are dealing with a classification problem. Because this is a binary classification task, a data point is considered *correctly* classified if the projection onto the label state has probability higher than 0.5. Given a parameter $\overline{\theta}$, a set of data points $\mathbf{x}$ and labels $\mathbf{y}$, the accuracy function is

$$acc(\overline{\theta},\mathbf{x},\mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n}1_{M(\overline{\theta},\overline{x}_i,y_i)>0.5} \qquad \text{(Equation 6)}$$

In the equation above, 1 represents the indicator function. A noticeable caveat to this choice is that as the accuracy improvements it gives *sparse* signals. That is, a classifier that correctly classifies all data points with 100% confidence is indistinguishable from one that merely gives 51% confidence under this metric. Indeed, it is desirable to have cost functions that are *continuous* and *differentiable* in classical neural networks. We now consider a common cost function that is often used in classical machine learning: *cross-entropy (CE) loss*.

$$CE(\overline{\theta},\mathbf{x},\mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n}1_{M(\overline{\theta},\overline{x}_i,y_i)>0.5}\log M(\overline{\theta},\overline{x}_i,y_i) \qquad \text{(Equation 7)}$$

We note that the CE loss attaches a log probability to the indicator function, which helps join the discontinuity when the probability shifts from below 0.5 to above 0.5. Lastly, we also consider a third cost function from,[27] *chi squared (χ^2) loss*.

$$x^2\ (\overline{\theta},\mathbf{x},\mathbf{y})\frac{1}{n}\sum_{i=1}^{n}(1 - M(\overline{\theta},\ \overline{x}_i,y_i))^2 \qquad \text{(Equation 8)}$$

Similar to the CE loss, χ^2 loss is also continuous and differentiable. However, we note two difference between the two loss functions: 1) the χ^2 loss converges faster to 0 as the correct classification probability gets close to 1; and 2) the χ^2 loss is capped at 1 for incorrect classifications whereas the CE loss is not bounded above. In practice, both χ^2 loss and CE loss lead to comparable performance as we can see from our simulation results in Figure S6. However, the rate at which CE (–) responds to changes in the parameter is higher and hence crucial to achieve optimal value in a shorter time. In the following, we discuss the basics of each of the classical optimizer that are compared in the main article.

Genetic optimizer

The Genetic Algorithm (GA) is a family of optimization algorithms inspired by the process of natural selection, emphasizing the survival of the fittest principle. According to a recent survey, the general form of GA commences with the initialization of a population

Algorithm 1. Genetic Algorithm for Quantum Classifier

```
Data: Population Size ← S.
Data: Convergence function, Converged(…), returns True if stopping criteria are reached
Result: Best solution, $\bar{\theta}_{best}$
1  Generate initial population of S chromosomes $\bar{\theta}_1, \bar{\theta}_2, \ldots, \bar{\theta}_S$;
2  Initialize iteration counter $t = 0$;
3  Compute the fitness value of each chromosome, $f(\theta_i)$;
4  while not Converged ($t, f(\theta_1), f(\theta_2), \ldots, f(\theta_S)$) do
5     Randomly select a few chromosomes to allow for survival to the next iteration;
6     Select one or more chromosomes from the current population based on fitness and hyper-parameters;
7     Apply crossover operation;
8     Apply mutation;
9     Replace the old population with the newly generated population;
10      Update $t \leftarrow t+1$;
11  return $\bar{\theta}$ with the highest fitness value;
```

(Y) comprising n randomly generated chromosomes. Following this, the fitness of each chromosome is evaluated, forming the basis for the selection process wherein two chromosomes, denoted as C1 and C2, are chosen based on their respective fitness values.[28]

To thoroughly investigate these dimensions, we will first present a high-level abstract algorithm tailored at the quantum classifier problem, and then provide an analysis of each of these optimization dimensions.

In the setting of the quantum classifier described above, we may define a skeleton GA algorithm Algorithm 1 by letting $\theta \in \mathbb{R}$ be an individual gene, and a vector of such genes, $\bar{\theta} \in \mathbb{R}^n$, be a chromosome. Our initial population will be comprised of a set of randomly generated chromosomes. The population is then evaluated and carries out reproduction according some fitness function *f*. We will also discuss the choice of *f* in detail.

Like other machine learning algorithms, GA has various hyper-parameters that we need to tune. Here are a few common ones we investigated.

1. Encoding Scheme of individual genes:
   - The authors in[28] list several commonly used encoding schemes for translating a problem into a GA-applicable one. They also suggest that value-based encoding schemes are usually preferred in neural networks for finding the optimal weights. Our problem is closely related to optimizing a neural network and we used value-based encoding scheme.
2. Selection criteria for chromosome reproduction:
   - The selection function in GA determines whether a particular chromosome will be selected to reproduce based on its fitness value. For the quantum classifier, we experimented with several selection techniques in simulation. We found that Steady-State Selection (SSS) is a superior selection method (see Figure S8) and this is what we used in practice. Other selection functions often lead to premature convergence in our problem setting.
   - We also note here that the selection function acts as a convergence pressure for the GA. And this can be thought of as a knob that we can tune to trade-off exploration and exploitation to converge to the best possible value given a training iteration constraint.
3. Crossover type to decide how off-springs inherit parents' genetic information:
   - Crossover type determines how off-springs are produced using the genetic information from 2 or more parents. We compared one-point, two-point and scattered crossover when 2 parents are selected for mating. We find that all three cross-over types offer comparable performance in our problem setting. And unlike selection function, no crossover type suffers significantly from premature convergence and scattered crossover seems to give slightly better performance (see Figure S8). This is what we used for our experiment.
4. Mutation function to decide how often and how much mutations occur:
   - In our simulation runs, we see various mutation probabilities significantly impacts convergence. With fixed mutation probability, GA performs the best with a mutation rate of 20% and gradually degrades with higher mutation values. This is most likely caused by an over-emphasis on the exploration of our parameter space rather than exploiting what we have learned already.

Algorithm 2. Gradient Optimization for Quantum Classifier

```
Data: Initial parameter $\bar{\theta}$
Data: Gradient Optimizer Function, Opt, returns False if converged
Result: Best parameter $\bar{\theta}_{\text{best}}$
1 Initialize iteration counter $t = 0$;
2 Function to compute metric of parameter, $f(\bar{\theta})$;
3 Initialize $\bar{\theta}_{\text{best}} = \bar{\theta}$;
4 Initialize $\bar{\theta}_{\text{new}} = \bar{0}$;
5 while $Opt(t, \bar{\theta}, f, \bar{\theta}_{\text{new}})$ do
6     Conditionally update best parameter, $\bar{\theta}_{\text{best}}$, to $\bar{\theta}_{\text{new}}$;
7     Update $\bar{\theta} \leftarrow \bar{\theta}_{\text{new}}$;
8     Update $t \leftarrow t+1$;
9 return $\bar{\theta}_{\text{best}}$;
```

- In classical deep learning, adaptive learning rate and weight decay are often employed as explicit knobs to decide how the rate of exploration-exploitation trade-off changes.[39] Borrowing this idea, we also introduced the concept of decaying mutation rate: $1_{p(\text{maskbase}^t)} * \delta * t^{\text{scale}}$. Here maskbase and scale are both hyper-parameters and t is the iteration number. δ is a uniform random variable between -0.5 and 0.5. We observe that this leads to better performance when compared to using a constant mutation rate as used for Figure S8.

5. Size of the population pool:
   - Population size is the most important factor in determining how well a GA converges. From Figure S8, we can see that lower population tends to converge early to unfavorable local minima and higher population explores for longer but eventually converges to better minima. We note that unlike the previous hyper-parameters, population size is directly related to amount of resource required. In practice, resource is scarce and our choice of population size was 50 during our GA experiments.

In addition to the results with genetic optimizers in the main article, we also include a couple of experimental runs with population sizes of 9. As we can see from Figure S9, this setup suffers from premature convergence due to low population size as shown in our simulation runs in sec. (*genetic-optimizer*). Next, we delve deeper into the use of gradient-based optimizer in NISQ devices, their pros and cons.

Gradient-based optimizer

Another class of popular algorithm (Algorithm 2) is gradient-based methods, which is an iterative optimization method that leverages first and higher order derivatives. These methods are very popular in classical machine learning algorithms as they are the *defacto* method used in optimizing deep neural networks. We explore two gradient-based methods in this paper. First is Stochastic Gradient Descent (SGD), which is a very popular method in the classical deep learning community. Second is a quasi-newton method, Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.

Stochastic Gradient Descent. Stochastic Gradient Descent (SGD) is a widely used optimization technique, especially prevalent in the training of machine learning models. It is an iterative method for optimizing an objective function with suitable smoothness properties (e.g., differentiable or sub-differentiable). The core principle of SGD is to perform the optimization by updating the parameters incrementally, using a subset of training samples at each step. This is in contrast to traditional gradient descent, which uses the entire data set to compute the gradient at each iteration. Mathematically, the update rule of SGD in the quantum classifier setting can be defined as:

$$\bar{\theta}_{t+1} = \bar{\theta}_t - \text{lr}\frac{1}{\text{Batchsize}} \sum_{\bar{x} \in \text{mini}-\text{batch}} \nabla f(\bar{\theta}_t, \bar{x})$$

where:

- $\bar{\theta}_t$ represents the parameter vector at iteration t.
- lr is the learning rate, a hyper-parameter that determines the step size at each iteration.
- $f(\bar{\theta}_t, \bar{x})$ is the average loss function computed for the mini-batch.
- $\nabla f(\bar{\theta}_t, \bar{x})$ denotes the gradient of the loss function with respect to the parameters.

By utilizing only a few training examples at each step, SGD significantly speeds up the optimization process, albeit at the cost of a more noisy convergence path. In simulation, we have found that SGD's convergence is significantly impacted by this noise and decided not to implement it for the experiment, which is an even noisier environment. We also found that even when we take the batchsize to be the entire training set, the rate of convergence is inferior to other optimizers and decided against running it in our experimental setup.

Quasi-Newton method. The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is a prominent iterative method for solving unconstrained nonlinear optimization problems. It uses second derivative to help guide the optimization path and takes a better convergence path when compared to gradient descent or SGD methods. It also belongs to quasi-Newton methods, a category of popular optimization techniques that approximate the Hessian matrix of second derivatives, which is a lot less expensive than computing the Hessian directly. In the context of optimization, the BFGS method seeks to find the minimum of a function f, where the update rule for each iteration k is given by:

$$\overline{\theta}_{k+1} = \overline{\theta}_k - \alpha_k B_k^{-1} \nabla f(\overline{\theta}_k)$$

The BFGS method approximates B_k, the inverse of the Hessian matrix, using the following update rule:

$$B_{k+1} = B_k - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}$$

where:

- $\overline{\theta}_k$ represents the parameter vector at iteration k.
- α_k is a scalar that typically satisfies the Wolfe conditions.
- $\nabla f(\overline{\theta}_k)$ is the gradient of the function f at $\overline{\theta}_k$.
- $s_k = \overline{\theta}_{k+1} - \overline{\theta}_k$ is the difference in consecutive parameter vectors.
- $y_k = \nabla f(\overline{\theta}_{k+1}) - \nabla f(\overline{\theta}_k)$ is the difference in consecutive gradients.
- B_k is the approximation to the inverse Hessian matrix at iteration k.

Gradient estimation. The BFGS method is renowned for its superior convergence properties. In order to leverage this algorithm, it remains to estimate the gradient at each desired $\overline{\theta}$ values. For classical machine learning problems, BFGS-based methods usually approximate derivatives using the finite difference approach. Ideally a step-size below 0.1 is needed to guarantee goodness of convergence. However, this step-size is hard to achieve in practice due to random noises in our system. We also attempted a different technique called *parametershift* to combat this issue.[31] This technique allows for the estimation of gradient at a particular parameter using arbitrary shifts in two directions. We fixed a shift size of $\frac{\pi}{2}$ in order to maximally leverage this technique and reduce the effect of random noise. We defer readers to sec. (algorithmic error in NISQ era) for detailed experimental error analysis.

Classical processor

The rising popularity of deep learning lies in its ability to solve more and more problems better than humans. To this end, we also attempted training the quantum classifier using a classical reinforcement learning (RL) agent. We formulated our problem as an RL problem and used the DQN algorithm,[40] with the following problem setup:

State: A vector of our classifier's parameter values.

Action: Increment, decrement, tiny increment, tiny decrement for each parameter. In addition, we have a reset action that either randomizes our parameter values for random initial value setup, or resets the parameter back to the initial value for fixed initial value setup. We note that these increments and decrements are fixed with some uniform noise.

Next State: The state we reach by taking an action.

Reward: Let A_i denote the accuracy at step i. The reward at step t is defined as $\max\left(0, \frac{A_t - \max_{j<t} A_j}{100 - A_0}\right)$.

With the above reward definition, we note that the agent is rewarded for finding higher accuracy values, but will not be penalized if it has to temporarily go through regions with low accuracy values. The overall architecture is shown in Figure S10. We used a densely connected neural network with 3 hidden layers of size 512. We used ReLU as our activation layer. The input size of our problem is vectors of length 16 and the action space is of size 65, using the above definition.

We attempted to train two different agents under this setup. A *random initial param* agent that always starts or resets with some randomized parameter values. A *fixed initial param* agent that picks a vector of 16 random parameter values at step 0 and reuses it for the entire training process. The agent is allowed to take 500 actions and its total reward is the sum of discounted rewards using under the standard DQN paradigm. Our replay buffer has up to 1,000,000 entries and our discount factor is 0.99. We used a learning rate of 1^{-4} and mini-batch size of 32. The exploration rate starts at 1 and decreases to 0.01 after 250,000 steps.

We can see from Figure S11 that the random agent learns very limited amounts of information while the fixed agent reaches accuracy values above 85% towards the end. We believe that this is due to the problem landscape as illustrated in the main text. The problem is filled with many local minima and therefore instead of learning good policies, the agent ends up memorizing nearby minima, which is easier to do when the initial value is fixed.

Algorithmic error in NISQ era

Let's return to our original problem to begin our error analysis journey. We will first consider the error accumulated from measuring a single quantum state according to the setup described in the *experimental model and method details* section. From there, we will build up the entire training dataset, and examine various types of errors and mitigation strategies. We conclude that with our current NISQ setup, our measurements are too noisy to be used for gradient-based optimizers.

Measuring a single quantum state

For this analysis we will take a concrete data point:

$$\overline{x} = \begin{bmatrix} 0.0976 \\ 0.4304 \end{bmatrix} \quad \text{(Equation 9)}$$

And we use a randomly initialized parameter $\overline{\theta}$:

$$\overline{\theta}^T = \begin{bmatrix} 0.1532 & 0.5374 & 2.3999 & -0.8025 & 0.3855 & 3.6122 & 1.2990 & 0.1235 & 1.3819 \\ 3.3182 & -2.4787 & 5.4758 & -1.4164 & 4.7989 & 0.5262 & 4.4542 & & \end{bmatrix} \quad \text{(Equation 10)}$$

We will use Ansatz-2C with $L = 4$. Thus we have:

$$|\phi\rangle = U_3(\overline{\theta},\overline{x})U_2(\overline{\theta},\overline{x})U_1(\overline{\theta},\overline{x})U_0(\overline{\theta},\overline{x})|0\rangle \quad \text{(Equation 11)}$$

$$= R_z(\theta_{14}x_0 + \theta_{15}x_1)R_y(\theta_{12}x_0 + \theta_{13}x_1)\ldots R_y(\theta_0 x_0 + \theta_1 x_1)|0\rangle \quad \text{(Equation 12)}$$

$$= R_z(0.2462)R_y(6.1721)\ldots R_y(1.9684)|0\rangle \quad \text{(Equation 13)}$$

Following the above, we may compute $|\phi\rangle = \alpha|0\rangle+\beta|1$ and deduce that $\alpha^2 = 0.663$ and $\beta^2 = 0.337$.

We will now see what $|\phi\rangle$ looks like in our experimental setup in Figure S12, where we performed repeated and independent measurements of 125 to estimate the value of β^2. Since we define $|0\rangle$ as the ground state and $|1\rangle$ as the excited state, the photon count will be higher (>12) when the qubit is in the excited state and lower ($\leq$12) when its in the ground state. We can see that for this particular data point, our measurement value of 0.344 is quite close to the theoretical value of 0.337.

Measuring the training dataset

In the previous section, we looked at how a single data point and parameter combination is measured using a qubit. We will now repeat this process for an entire training dataset.

We can learn a few things from Figure S13. First, due to the natural decoherence of the qubit, the experimental measurements gradually tilt towards the $y = 0.5$ line. In this particular case, the average amount of decoherence results in the regression line moving from $y = x$ to $y = 0.106 + 0.759x$. This can be classified as a type of systematic error caused by the environment. Second, we see the dotted points scattered around the regression line. This is a result of random uncertainty from the experimental setup. This is a combination of systematic errors from instruments and random error from environments. We will now attempt two separate error mitigation techniques to overcome the errors we identified above.

Mitigation of systematic error

A qubit in the real world experiences quantum decoherence overtime. This error can be mitigated post-measurement at the algorithmic level by defining the following matrix:

$$M = \begin{bmatrix} P(|0\rangle\|0 & P(|1\rangle\|0 \\ P(|0\rangle\|1 & P(|1\rangle\|1 \end{bmatrix} \quad \text{(Equation 14)}$$

where $P(|0\rangle\|1$ represents the probability of measuring $|0\rangle$ when our theoretical prediction is $|1\rangle$. On our experimental setup, we prepare a sequence of dummy gates whose total gate time is the average of that of our training dataset. But those dummy gates produce end quantum states of either $|0\rangle$ or $|1\rangle$. Here is our measured matrix M:

$$M = \begin{bmatrix} 0.76 & 0.24 \\ 0.16 & 0.84 \end{bmatrix} \quad \text{(Equation 15)}$$

We then apply M^{-1} to our measurements in Figure S13 and obtain the new error mitigated result shown in the same figure presented in blue. Comparing figures in (S13) shows that applying M^{-1} to the experimental values helps mitigate systematic errors of the environment. The gradient of the regression line is closer to 1, and both the mean and standard deviation of the residual errors improved.

Mitigation of random error

Figures in (S13) suggest that the random errors of the experimental measurements are fairly normally distributed. If we follow this assumption, we expect our errors to decrease following $\frac{1}{\sqrt{N}}$, where N is the number of repetitions.

From Figure S15 we can see that the standard deviation of the residual error decreases proportionally to $\frac{1}{\sqrt{N}}$. This trend ends after 750 repetitions, where we start to observe random errors that do not scale with the number of repetitions. We refer to this an unknown or residual error. And this residual error of around 0.006 represents the best experimental bound we can achieve under our current experimental setup.

Finite-difference based gradient optimizers

In this sub-section, we examine the practicality of finite difference methods as our gradient-based optimizer. We see from Figure S16 that our measured gradients using step sizes of 0.5 and 0.1 are all overridden by our noise, with the exception of the gradient of accuracy when step size is 0.5. We proceeded with this setup in our experiment noting two caveats. Caveat 1 is that even under this setup, the signs of our gradients are sometimes on the wrong side, making it very difficult for the algorithm to improve. Caveat 2 is that the step size of 0.5 is much too large for any gradient-based algorithm to work well, as reflected in our experimental results.

In classical machine learning, we know that the step size should be chosen to be as small as possible for gradient-based optimizers to work well. Rigorously, the mathematical definition of gradient exists only as some small ϵ tends to zero. Fortunately, for our particular type of quantum circuit, we may also employ the *parameter-shift rule*[41] to compute the gradients. We will explore this in the next section. Additionally, we have conducted numerical experiments to demonstrate the robustness of the genetic algorithm (GA) over gradient-based methods. In Figure S14, we compare the performance of a typical GA solver against the l-BFGS-b solver, using various step sizes for the estimation of finite difference gradient. Although the l-BFGS-b solver performs well in noise-free environments with smaller step sizes (see Figures S14B and S14C), its performance deteriorates significantly with the introduction of noise. This behavior is expected as the absolute values of the gradients are comparable in magnitude to the noise levels.

In Figure 5, we further analyze the performance of the GA solver and the l-BFGS-b solver under Gaussian noise with a standard deviation of 0.05, consistent with our experimental setup. The noise-free results are presented alongside the noisy simulation results for comparison. Notably, the l-BFGS-b solver exhibits a significantly larger performance gap between noise-free and noisy conditions compared to the GA solver, highlighting the latter's robustness in noisy environments.

Parameter-shift rule and limitation of gradient-based algorithms

Referring to Figure S17, we see the theoretical gradients of various optimizers using our given $\overline{\theta}$ of length 16. The accuracy gradient is zero everywhere, matching our understanding that it is a *sparse* signal (see sec. *cost-functions*). On the other hand, the gradients of cross-entropy and chi-squared values are non-zero but the magnitude of their values are extremely small - orders of magnitude smaller than the residual standard deviation from sec. (mitigation of random error). This makes all gradient-based methods undesirable candidates for optimization.