RESEARCH-ARTICLE

AccelerQ: Accelerating Quantum Eigensolvers with Machine Learning on Quantum Simulators

AVNER BENSOUSSAN, King's College London, London, U.K.

I am a PhD Candidate in Computer Science at King's College London. My research focuses on understanding faults in Hybrid Quantum–Classical architectures and on developing foundational methods for robust and reliable quantum software. In particular, I investigate applications of quantum information theory to the testability and verification of quantum and hybrid software architectures.

ELENA CHACHKAROVA, King's College London, London, U.K.

KARINE EVEN-MENDOZA, King's College London, London, U.K.

SOPHIE FORTZ, King's College London, London, U.K.

CONNOR LENIHAN, King's College London, London, U.K.

Published: 09 October 2025
Accepted: 12 August 2025
Received: 26 March 2025

AccelerQ: Accelerating Quantum Eigensolvers with Machine Learning on Quantum Simulators

AVNER BENSOUSSAN, King's College London, United Kingdom
ELENA CHACHKAROVA, King's College London, United Kingdom
KARINE EVEN-MENDOZA, King's College London, United Kingdom
SOPHIE FORTZ, King's College London, United Kingdom
CONNOR LENIHAN, King's College London, United Kingdom

We present `AccelerQ`, a framework for automatically tuning *quantum eigensolver* (QE) implementations–these are quantum programs implementing a specific QE algorithm–using machine learning and search-based optimisation. Rather than redesigning quantum algorithms or manually tweaking the code of an already existing implementation, `AccelerQ` treats QE implementations as black-box programs and learns to optimise their hyperparameters to improve accuracy and efficiency by incorporating search-based techniques and genetic algorithms (GA) alongside ML models to efficiently explore the hyperparameter space of QE implementations and avoid local minima.

Our approach leverages two ideas: 1) train on data from smaller, classically simulable systems, and 2) use program-specific ML models, exploiting the fact that local physical interactions in molecular systems persist across scales, supporting generalisation to larger systems. We present an empirical evaluation of `AccelerQ` on two fundamentally different QE implementations: ADAPT-QSCI and QCELS. For each, we trained a `QE predictor` model, a lightweight `XGBoost` Python regressor, using data extracted classically from systems of up to 16 qubits. We deployed the model to optimise hyperparameters for executions on larger systems of 20-, 24-, and 28-qubit Hamiltonians, where direct classical simulation becomes impractical. We observed a reduction in error from 5.48% to 5.3% with only the ML model and further to 5.05% with GA for ADAPT-QSCI, and from 7.5% to 6.5%, with no additional gain with GA for QCELS. Given inconclusive results for some 20- and 24-qubit systems, we recommend further analysis of training data concerning Hamiltonian characteristics. Nonetheless, our results highlight the potential of ML and optimisation techniques for quantum programs and suggest promising directions for integrating software engineering methods into quantum software stacks.

CCS Concepts: • **Software and its engineering** → *Application specific development environments*; *Search-based software engineering*.

Additional Key Words and Phrases: Quantum Computing, Quantum Program Analysis, Optimisation, Machine Learning, Search-based Software Engineering, Genetic Algorithms

ACM Reference Format:
Avner Bensoussan, Elena Chachkarova, Karine Even-Mendoza, Sophie Fortz, and Connor Lenihan. 2025. AccelerQ: Accelerating Quantum Eigensolvers with Machine Learning on Quantum Simulators. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 354 (October 2025), 31 pages. https://doi.org/10.1145/3763132

Authors' Contact Information: Avner Bensoussan, King's College London, London, United Kingdom, avner.bensoussan@kcl.ac.uk; Elena Chachkarova, King's College London, London, United Kingdom, elena.chachkarova@kcl.ac.uk; Karine Even-Mendoza, King's College London, London, United Kingdom, karine.even_mendoza@kcl.ac.uk; Sophie Fortz, King's College London, London, United Kingdom, sophie.fortz@kcl.ac.uk; Connor Lenihan, King's College London, London, United Kingdom, connor.1.lenihan@kcl.ac.uk.

ACM 2475-1421/2025/10-ART354
https://doi.org/10.1145/3763132

1 Introduction

Modern *Noisy Intermediate-Scale Quantum* (NISQ) devices represent the current state of *Quantum Computing* (QC). They operate with a limited number of qubits that are prone to errors due to decoherence and imperfect control, and they lack the ability to perform fault-tolerant computations due to the inability to sustain deep circuits required to demonstrate quantum advantage [2, 12, 72]. Quantum simulators have become essential tools due to their accessibility, cost-efficiency, deterministic behaviour, and seamless integration into classical workflows. These simulators enable researchers to prototype, optimise, and validate quantum algorithms without immediate access to quantum hardware, to prepare for the future deployment on more reliable quantum hardware.

One of the most promising applications of QC lies in simulating quantum systems for chemistry and materials. In the 80s, Feynman originally envisioned QC as a means to simulate quantum matter more efficiently [31], and breakthroughs could accelerate advances in chemical production, materials design, drug discovery, and, more recently, AI, healthcare, and finance [1, 43, 44, 51, 54, 56, 70]. Classical computational methods, however, struggle with these problems because representing and analysing quantum systems requires resources that grow exponentially with system size. QC offers an avenue to accelerate the process of simulation by representing and manipulating a quantum state using polynomial resources.

Achieving optimisation in quantum computing (QC), outperforming classical methods, remains a challenge in the NISQ era. While optimisation is a core concern in both classical and quantum systems, quantum programs are not just classical code with quantum platform libraries' invocations: they implement unitary, reversible dynamics with stochastic measurements and run on noisy, NISQ devices. Consequently, optimisations can yield no benefit or degrade accuracy ([22, 38, 50] and as discussed in §8). Algorithmic frameworks and compiler-level tools, such as transpilers and circuit synthesis or reduction methods [38, 53, 59, 98, 99], support optimisation, but, in practice, achieving effective and reliable optimisation is a challenge [85, 99]. Optimisation targets circuit width (number of qubits), depth (gate layers or terms), specific gate parameters such as rotation angles or fidelity [59, 98, 99], further approaches utilise *machine learning* (ML) [53] or user annotations at the program level [38]. Further, directly optimising quantum implementations (*i.e.* programs that implement quantum methods and algorithms) is especially critical for near-term applications, where resource constraints are tight and execution time on real hardware is both limited and costly. One essential aspect of this optimisation is hyperparameter tuning, adjusting key algorithmic parameters such as gate configurations, gradient estimation methods, stopping criteria, resource allocation, and truncation thresholds in approximations. While the problem of hyperparameter optimisation has been extensively studied for classical algorithms [42, 103], it has also gained interest in the context of quantum algorithms, particularly in quantum ML [19, 42, 61].

This work aims to optimise constants in quantum algorithm implementations, formulating it as a hyperparameter optimisation problem. We investigate a class of quantum implementations known as Quantum Eigensolvers (QEs). A key problem in quantum chemistry and materials modelling is finding a system's lowest energy state (ground state), which determines properties such as reaction rates, spin interactions, and material stability. This importance has driven the development of various quantum algorithms for the task, including the Variational Quantum Eigensolver (VQE) [71, 87], ADAPT-VQE [36], and Quantum Phase Estimation (QPE) [49, 66], with corresponding libraries and interfaces already integrated into quantum platforms, as discussed in §3.1.

Our Contribution. We investigate the optimisation of quantum implementations of *Quantum Eigensolvers* (QE) designed to find the eigenvalues and eigenvectors of a given Hamiltonian system and, in particular, to calculate its lowest eigenvalue. We explore two QE implementations: (1) the *Adapt Quantum-Selected Configuration Interaction* (ADAPT-QSCI) [48, 77] and (2) the *Quantum Complex*

Exponential Least Squares (QCELS) [18, 24] eigensolvers. Our approach integrates SE and ML to enhance performance and accuracy under hardware-like constraints (*e.g.* limited memory, fixed shot budgets) utilising quantum simulators, thereby mimicking quantum hardware's limitations.

We combine these two strengths within a *search-based software engineering* (SBSE) optimisation framework [40]. **SE** enables code-level validation of hyperparameter suggestions, ensuring they are physically plausible and correctly integrated into QE implementations (*e.g.* propagating proposed values through actual executions and inspecting correctness beyond the final output). **ML** brings the ability to generalise from prior optimisation runs. Specifically, we utilise (i) Genetic Algorithm (GA) to explore the hyperparameter space via mutation and crossover, and (ii) ML to predict promising regions from training on small Hamiltonians ($\leq$ 16 qubits). This strategy avoids local minima, accelerates convergence, and, by incorporating Hamiltonian structure, provides a richer, physically informed search space. It addresses a second challenge: solving systems above 16 qubits are often prohibitively expensive or infeasible in the current NISQ era [62].

By training on small, classically simulable systems, ML models can capture correlations between Hamiltonian properties and effective hyperparameters under a given QE implementation. It is feasible because local physical interactions in molecular systems persist across scales (*e.g.* a small hydrogen chain appears identically as a subsystem in a larger one), allowing the model to learn from these recurring structural building blocks. Nonetheless, the framework is implementation-agnostic: it operates on any Hamiltonian and takes the QE implementation Python and quantum libraries code as input. The only implementation-specific components are a testing mechanism to detect physically invalid hyperparameter values and the extraction of all tunable constants, which can differ between implementations, even for the same quantum algorithm.

We developed `AccelerQ` to optimise QEs' hyperparameters. An ML model, `QE predictor`, trained on evaluations of small systems per QE implementation, was used to generalise to larger systems. In our experimental evaluation, we trained `AccelerQ` on *small systems*: systems up to 16 qubits and predicted the hyperparameters for *larger systems*: 20-, 24-, and 28-qubit systems. All 20–28 qubit test systems were fully unseen during training. Optimisation was then performed per QE implementation and target Hamiltonian, thus each larger system receives its own optimised hyperparameters. We compared these against the default hyperparameters, which remain fixed regardless of the QE's input. Results show clear improvements for complex systems, particularly at 20 and 28 qubits, with minimal or no gain on simpler systems with fewer Hamiltonian terms.

Figure 1 illustrates the architecture of `AccelerQ` and highlights the interplay among its core components, which directly align with the key contributions of this paper:

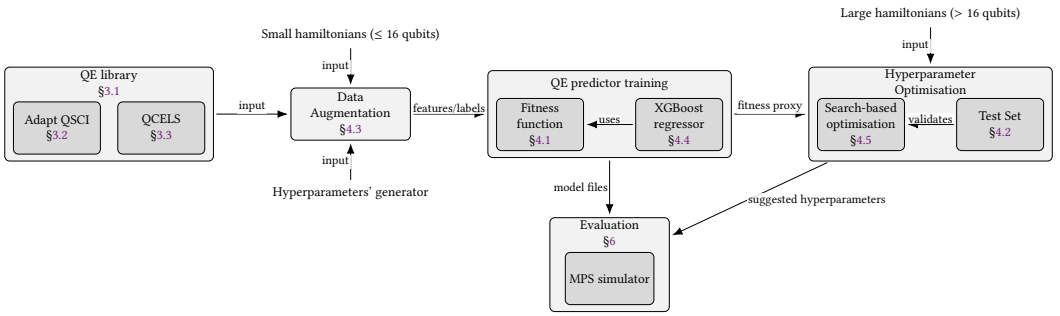


Fig. 1. `AccelerQ` at a glance: inputs from the QE library feed data augmentation and training (*QE predictor*), which serves as a fitness proxy for hyperparameter optimisation; suggestions are evaluated on a simulator.

- **Formulating the Optimisation Problem (§2)**: We describe the fitness function of our optimisation problem as a hyperparameter optimisation problem of a given QE and a Hamiltonian.
- **A General Framework for QE Optimisation (§4)**: We propose a search-based framework combining ML and GA to optimise QE implementations' hyperparameters at the level of QE implementation and a problem Hamiltonian.
- **Scalable Learning from Small Quantum Systems to Apply on Larger Systems (§4)**: We introduce a methodology that learns patterns from small, classically simulable quantum systems (≤ 16 qubits) and applies this knowledge to optimise simulations of larger systems (20–28 qubits).
- **`AccelerQ` Implementation (§5)**: We implemented our approach in a new tool, `AccelerQ`, which treats the QE implementations as a black box, requiring no internal modification.
- **Empirical Evaluation on Two QE Implementations (§6, §7)**: We evaluated `AccelerQ` on two use cases (ADAPT-QSCI and QCELS) across 16 Hamiltonian systems.
- **Manual Analysis and Validation (§8, §9)**: We further investigated the correctness of the results manually, beyond comparing the reference result, ensuring reported results are sensible and stem from a valid computation of the lowest eigenvalue and not by chance.

We empirically evaluated our contributions through five research questions (*RQ*s).

RQ_1 *How does* `AccelerQ` *affect hyperparameter values in QE implementations compared to their default settings?*

RQ_2 *Can* `QE predictor` *models trained on smaller systems make useful predictions for optimal hyperparameters generalise across system sizes?*

RQ_3 *To what extent can* `AccelerQ`*'s optimisation of hyperparameters accelerate and improve the efficiency and accuracy of QE implementations in terms of system size?*

Via an ablation study, in RQ_4, we compare the performance of `AccelerQ` against a weaker variant of it, to assess whether the additional effort introduced by genetic algorithms and code-level validation of hyperparameter suggestions, *i.e. a set of tests*, yields meaningful improvements.

RQ_4 *How scalable is each configuration, in terms of iterations and error rate, when applied to QE implementations for Hamiltonian systems with increasing qubit number and complexity?*

RQ_5 *To what extent does* `AccelerQ` *affect the variance of QE results (error, iterations, and final energy) across Hamiltonians of the same size?*

RQ_5 goes beyond achieving the lowest scores: it examines the optimisation process's stability, reproducibility, and threats to validity. The *RQ*s evaluate each contribution's impact, first the `QE predictor` model, then the *genetic algorithm* (GA) guided by a test set, on hyperparameter optimisation. We assess `AccelerQ` by comparing it against the baseline (the QE's default hyperparameters) and two variants of `AccelerQ`: (1) only ML, and (2) combining ML with a GA guided by a test set.

2 Quantum Eigensolvers as Optimisable Software Components

Quantum eigensolvers (QEs) are quantum algorithms designed to approximate primarily the lowest energy eigenvalue of a physical system and the corresponding eigenstate. A *quantum system* is defined by a *Hamiltonian*, an operator that describes how the system evolves and encodes its total energy.

Estimating energy levels of large quantum systems remains computationally demanding: current NISQ hardware suffers from noise that scales exponentially with the size of the studied system. Hence, the circuits we are investigating (≥ 20 qubits) would produce unusable outputs on quantum hardware available today. Simulating such circuits classically eliminates noise, but requires exponentially growing resources, making new quantum-based techniques and their optimisation essential for estimating energy levels in larger systems.

Fig. 2. Overview of QE algorithms, with the grey-dashed area indicating where automated software engineering and machine learning can enhance outcomes. *Initial State* is the Ansatz or Reference State; *Hyperparameters* are numerical/boolean data and differ between implementations of QE; *Problem Hamiltonian* is an operator or function that represents the total energy of a system; and *Improved* E_{est} is the improved (low-error) lowest eigenvalue approximation, possibly including also the lowest eigenstate.

QE implementations (§3.1) combine a series of operations, *e.g.* quantum gates, on the quantum simulator or hardware and measurements of the state of the system to either prepare a representation of the eigenstate of the Hamiltonian or directly evaluate its eigenvalue. QEs take as input a problem Hamiltonian, an initial state[1], and a set of hyperparameters as input (see Figure 2). They return an estimate of the lowest eigenvalue with performance depending on the specific QE implementation, the properties of the Hamiltonian, and the suitability of the chosen hyperparameters in the context of the input Hamiltonian. This can be formulated as a minimisation optimisation problem:

$$f_{\text{QE}}(\boldsymbol{\theta}; \hat{H}) = \left| E_{\text{est}}(\boldsymbol{\theta}; \hat{H}) - E_{\text{true}}(\hat{H}) \right| \tag{1}$$

thus **1)** $\hat{H}$ is the input Hamiltonian, **2)** θ denotes the QE hyperparameters, **3)** $E_{\text{est}}(\boldsymbol{\theta}; \hat{H})$ is the estimated lowest eigenvalue from the QE implementation, and **4)** $E_{\text{true}}(\hat{H})$ is the true ground state energy (computed classically or known analytically). This absolute error defines the *fitness function* and is used to optimise hyperparameters for accuracy. When E_{true} is unavailable, the error rate percentage likely cannot be computed due to complexity of the Hamiltonian system; however, for variational methods, a more accurate prediction corresponds to a lower (*i.e.* more negative) estimated energy, assuming this property is correctly encoded in the QE implementation. If the implementation is faulty, trivially, optimisations are not expected to yield correct results.

Research Problem. This work aims to improve the accuracy of quantum eigensolver (QE) implementations through automated optimisation. The areas shaded in grey in Figure 2 are those parts we can control and optimise with automated software engineering methods. We operate under the constraint that the QE implementation is treated as a black box[2]. Consequently, we focus on identifying hyperparameter configurations that improve estimation accuracy for a given input Hamiltonian, and defer the optimisation of initial state preparation to future work.

`AccelerQ` takes as input the source code of a QE implementation (in our case, written in Python), along with manually identified terms and their associated data types that are expected to influence performance (currently through manual inspection, with plans for automation). Based on these parameters, `AccelerQ` samples the QE's behaviour in a black-box manner using small input Hamiltonians to train a `QE predictor` model. This model is then used to automatically adjust the relevant constants to optimise performance for larger input systems as a preprocessing step before execution. Incorporating the Hamiltonian into the workflow or modelling of quantum programs, although less conventional than the circuit model, opens up further opportunities for program analysis, particularly in simulation-driven domains like quantum chemistry.

[1]Typically the Hartree-Fock state in chemistry applications [84].
[2]This is partly because we aim to present a general solution that takes a QE implementation as input, rather than re-writing its internal code manually, and partly because constructing or improving a QE implementation remains a significant challenge in its own right [76, 77], arguably falling within the domain of quantum physics research.

3 Background

Our approach combines quantum computing (§3.1), exemplified on two QE used in our evaluation (§3.2 and §3.3), with machine learning techniques applied to software engineering (§3.4).

3.1 Quantum Implementations and Quantum Eigensolver

We refer the reader to [66] for background on quantum gates, circuits and Hamiltonians. We focus here on quantum computational models and QE implementations. Quantum implementations are programs written in languages such as Python, Java, or C/C++, using dedicated libraries provided by *quantum computing platforms*. Many quantum platforms provide ready libraries, interfaces or templates for QE implementations, such as the standard quantum chemistry libraries in Qiskit [73] and PennyLane [96], whilst others have external packages like Cirq [74] and Braket [4]. Specifically, a VQE implementation is included in the Qiskit Algorithms library. Qiskit also defines general-purpose eigensolver interfaces (not limited to VQE-style algorithms), such as Eigensolver and MinimumEigensolver. Nonetheless, quantum platforms can support QE execution via plugins or external libraries, even if QE functionality is not included natively, for example, with `OpenFermion-Cirq`, `pennylane-braket`, and `pennylane_qiskit`, *e.g.* [17] (subsection 7.4.3).

Quantum eigensolver implementations can be broadly categorised by how they interact with the input Hamiltonian. Most commonly, the Hamiltonian is translated into a parametrised quantum circuit, as in variational and phase estimation algorithms under the quantum circuit model [66, 71]. Alternatively, purely classical eigensolvers may operate directly on the Hamiltonian through exact diagonalisation, but are limited by unfavourable scalability [20]. A third category includes quantum computational models that act directly on the Hamiltonian without circuit translation, such as QCELS, adiabatic quantum computation, and quantum annealing, which represent different but less common non-circuit-based approaches to quantum computing [29, 46]. In general, these methods are less suited to NISQ devices as they rely on continuous-time evolution or hardware-specific requirements that are difficult to implement on noisy gate-based hardware.

In §6, we consider two representative examples: one translates the Hamiltonian into a quantum circuit (§3.2), and another operates directly on the Hamiltonian without circuit decomposition (§3.3). In this paper, we treat each QE as a black box, with its internal lowest-energy minimisation objectives defined in [18, 63]. Our optimisation instead targets the prediction error defined in Equation 1. Consequently, we provide a high-level description of each QE (in §3.2 and §3.3) and refer the reader to the original publications for full mathematical details.

3.2 ADAPT-QSCI Algorithm

The *Quantum-selected configuration interaction* (QSCI) method is a quantum chemistry algorithm that calculates molecular electronic structures in an intelligently chosen subspace, enabling larger systems to be studied on modern NISQ devices [48]. An exact calculation in the full Hilbert space, even when constrained by symmetries, requires a high computational cost and memory usage infeasible for large Hamiltonians. QSCI reduces the computational space by selecting the subspace consisting of only the computational basis states (aka configurations in quantum chemistry) with the highest weight in some pre-chosen input state prepared on a quantum computer. Hamiltonian diagonalization is in the selected R_k dimensional subspace $S_k = \text{span}\{|r_1^{(k)}\rangle, \ldots, |r_{R_k}^{(k)}\rangle\}$ [48]. QSCI uses a quantum computer only to generate the subspace via sampling. The subsequent calculation to output the ground-state energy is executed on classical computers. This is feasible on classical machines because QSCI reduces the subspace dimensionality.

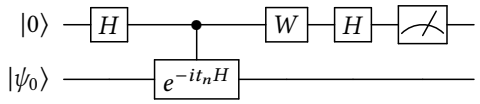


Fig. 3. QCELS circuit [24]. W gate represents an optional $S^\dagger$ gate to calculate the imaginary part of the result, and if removed, the real part is calculated.

The Adaptive Construction of Input State for Quantum-Selected Configuration Interaction (ADAPT-QSCI) algorithm [63] iteratively uses QSCI to construct the input state for the next QSCI iteration. At each step, a subspace is selected by measuring the quantum state from the previous step. ADAPT-QSCI chooses the next quantum gate to add to the input state from a predefined set of multi-qubit Pauli operators $\mathbb{P} = \{P_1, \ldots, P_T\}$ by calculating the gradients, $h_j = \langle c_k | i[H, P_j] | c_k \rangle$[3], in the subspace S_k. The Pauli operators are generators of rotation gates. The optimal rotation angle of the gate (the angle which lowers the energy of the state the most) is also found classically in the selected subspace S_k. The algorithm is similar to ADAPT-VQE [36], but differs in how the next gate and rotation angle are chosen. Whereas ADAPT-VQE selects them directly through quantum measurements, ADAPT-QSCI computes them classically within a subspace informed by measurements of the previous quantum state while utilising quantum computing in the state preparation and measurement steps.

3.3 QCELS Algorithm

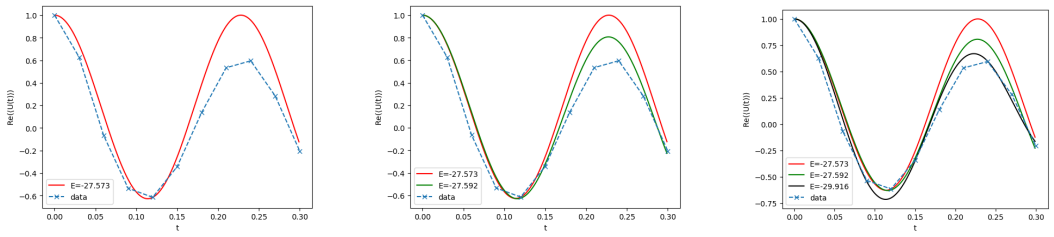
We utilise the Quantum Complex Exponential Least Squares algorithm (QCELS) [24]. NISQ algorithms typically prepare an ansatz state and measure it in a Pauli basis [66], whereas QCELS uses a controlled time-evolution unitary. It avoids the optimisation issues that hinder large-scale variational NISQ algorithms [15, 57] and has a low enough circuit depth to be likely suitable for early error-corrected quantum computers, making it well-suited for problems with more qubits than NISQ algorithms can handle.

QCELS takes a reference state $|\psi_0\rangle$ and evolves it by the time evolution operator $U(t) = e^{-iHt}$, where H is the Hamiltonian system. The time evolution operator is enclosed within a Hadamard test (as depicted by Figure 3), see [66] for an introduction to quantum circuits. This circuit measures the overlap between the time-evolved state $U(t)|\psi_0\rangle$ and the initial reference state $|\psi_0\rangle$. If the reference state is not exactly the ground state, the resulting expectation value as a function of n is $Z_n \approx \langle\psi_0|U(t_n)|\psi_0\rangle = \Sigma_i p_i e^{-iE_i t_n}$ where $p_i = |\langle\phi_i|\psi_0\rangle|^2$ is the probability of measuring the eigenstate $|\phi_i\rangle$ of the Hamiltonian. Thus, if a reference state with good overlap with the true ground state is known, we can apply the time evolution operator within a Hadamard test N times and fit to the resulting complex exponential. We implemented the time-evolution operator using a first-order Trotter–Suzuki expansion, truncating the Hamiltonian to the terms with the largest coefficients to reduce gate count. We fit the resulting data with a function composed of a sum of three complex exponentials (Equation 2), to partially account for the inexactness of the reference state.

$$f_{\text{fit}}^{(3)} = r_1 e^{-i\theta_1 t} + r_2 e^{-i\theta_2 t} + (1 - r_1 - r_2) e^{-i\theta_3 t}, \tag{2}$$

In Figure 4, we present results for test data collected from an orbital rotated Hubbard Hamiltonian on a 28-qubit example (in blue dashed line), with a sequential improvement towards the correct answer as we increase the number of frequencies in the fit: first fitting in red, second fitting in

[3] $|c_k\rangle$ is a state corresponding to classical vector $c_k = \sum_{l=1}^{R_k} (c_k)_l |r_l^{(k)}\rangle$, and $i[H, P_j]$ is calculated through projecting onto the subspace S_k and evaluating the expectation classically using the classical vector c_k.[63].



(a) $f_{\text{fit}}^{(1)}$ (b) $f_{\text{fit}}^{(2)}$ (c) $f_{\text{fit}}^{(3)}$

Fig. 4. QCELS results from orbital rotated Hubbard Hamiltonian, 28-qubit sample data, various parameters.

green and third fitting in black, showing an improvement throughout the process of a minimisation problem.

3.4 Software Engineering Optimisation Methods

Search-based software engineering (SBSE) techniques and genetic algorithms (GA) have been widely applied to diverse SE problems such as code and requirements optimisation [34, 40, 41, 55]. GA aims to find the near-optimal solution iteratively, ending when a stopping condition is met, *e.g.* reaching the maximum iterations. Commonly, such approaches start with a set of candidate solutions, each evaluated using a fitness function. At each iteration, the best solutions are selected based on fitness, and variation is introduced through mutation (*e.g.* bit flips) and crossover to explore the search space and produce potentially better offspring. To avoid premature convergence to local optima, diversity can be maintained by adding noise or removing a portion of the candidates. When direct evaluation of the fitness function is computationally expensive or infeasible, the fitness function can be approximated for large Hamiltonians with machine learning (§4).

The Gradient Boosting technique [32] is a supervised learning method suitable for regression and classification. It efficiently handles large amounts of data, real numbers and datasets with many features. In this work, we use it to train `QE predictor` in §4. The resulting `QE predictor` model enables high-precision predictions on large datasets, effectively avoiding flat predictions from scaling issues or oversimplified outcomes caused by insufficient data relative to the number of features.

4 AccelerQ

This section details the methodology of `AccelerQ`: data augmentation, predictor training, and search-based optimisation. These are not contributions in isolation but, together, instantiate our core contribution: embedding SE and ML into the optimisation loop of QE implementations to reduce error under hardware-like constraints (§6). In our evaluation, we applied this process to two QE implementations (§3.2, §3.3) and summarised the findings in §7.

More concretely, `AccelerQ` aims to enhance quantum implementation performance through hyperparameter optimisation. We manually examine the QE implementation to extract a hyperparameter set[4] and formulate a hyperparameter optimisation problem. `AccelerQ` utilises SSBS and ML methods to optimise a QE implementation via tuning of the hyperparameter vector θ, for

[4] *i.e.* some expressions are extracted from the inner code, like the terms governing the Hamiltonian representation, while others are already part of the parameters of the original problem, such as the number of shots.

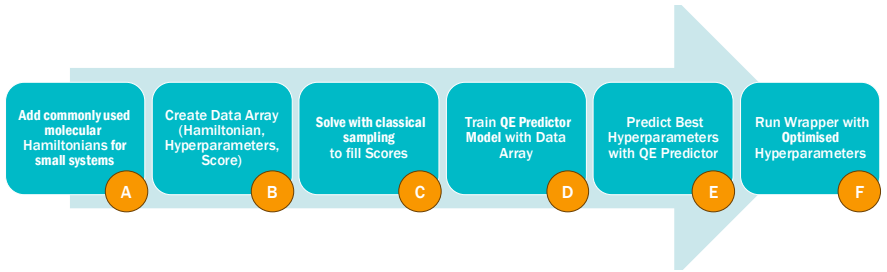


Fig. 5. AccelerQ process: Classical pre-processing, QE predictor model training on smaller systems and generalisation on bigger systems

which an exhaustive evaluation is computationally infeasible, particularly for large Hamiltonians. Selecting optimal parameters can reduce the number of shots (measurements) and improve the accuracy of ground-state energy predictions.

Figure 5 illustrates our hyperparameter optimisation algorithm for quantum problems, which follows the phases below.

Prepare the QE implementation for tuning. Given an *implementation:* a QE implementation to optimise, we identify tunable constants in the QE implementation (some already exposed as parameters, others moved out of the code) and write tests to validate hyperparameter values. Given $\boldsymbol{\theta}$ and a QE's test set, we can now define the *Hyperparameters Generator: generator*, a function to generate valid hyperparameters for a given QE implementation. Each implementation requires its own generator.

This phase defines generator, $\boldsymbol{\theta}$ (§4.1) and a QE's test set (§4.2).

Data extraction & augmentation. (Ⓐ, Ⓑ, Ⓒ Figure 5) Given an implementation and a *generator* we sample solutions from *few-qubit systems:* a collection of small-Hamiltonian system problems (16 qubits or fewer) from open-source/benchmark datasets. We run the QE implementation in *classical simulation mode:* a statevector simulator[5], to generate training data for the input implementation, where $E_{\text{est\_classical}}$ is the estimation of a QE's lowest eigenvalue computations in classical simulation mode. Thus can compute, $y = E_{\text{est\_classical}}(x)$, where:

- Inputs (X): the tuple $x = (\boldsymbol{\theta}; \hat{H})$, which is the compressed/flattened Hamiltonians and the hyperparameter vectors.
- Outputs (Y): exact energy values, y, (computed classically).

Note that, AccelerQ uses the four input sets (implementation, Hamiltonian systems, statevector and *generator* detailed above) to extract data arrays classically, based on the hyperparameter format (θ) and the specific hyperparameter ranges of that implementation (*i.e.* the QE's test set, which can include tests as the maximum number of iterations is an integer, whereas the Hamiltonian coefficient cut-off is a float), which are encoded into the hyperparameters *generator*.

This phase defines the training dataset saved as feature–label pairs (X,Y) (§4.3).

Train QE predictor. (Ⓓ, Figure 5) Using the feature–label pairs (X, Y) dataset from the data extraction & augmentation phase, AccelerQ trains an XGBoost model on the dataset to predict ground-state energy (Ys) from {hyperparameters, Hamiltonian} pair (Xs). We save it as the QE predictor model for the input QE implementation to optimise.

[5] *e.g.* see https://www.epcc.ed.ac.uk/whats-happening/articles/energy-efficient-quantum-computing-simulations.

This phase defines `QE predictor` *for the input implementation (§4.4).*

Deploy `QE predictor` **for optimisation. (Ⓔ, Figure 5)** Given an implementation, its `QE predictor` and *generator*, and a large system (Hamiltonian of more than 16 qubits), `AccelerQ` generates a prediction of optimal hyperparameters, using `QE predictor` as our fitness function (*i.e.* $f_{\text{QE}}(\boldsymbol{\theta}; \hat{H})$, Equation 1), *generator* with mutation and crossover operators and a GA feedback loop to refine the initial suggestions. `AccelerQ` iteratively proposes candidate hyperparameters, `QE predictor` predicts their performance, and the best candidates are propagated for further mutations.

This phase defines the optimised hyperparameters for an implementation and a Hamiltonian (§4.5).

`AccelerQ` provides tailored hyperparameter recommendations for executing an input QE implementation, customised for each specific Hamiltonian and QE implementation (Ⓕ, Figure 5). We compare each QE implementation's default hyperparameters against those optimised by our framework with an MPS simulator to assess the quality of the prediction in the evaluation §6.

Justification: Direct evaluation of the lowest eigenvalue given a Hamiltonian system is (1) costly to compute on quantum hardware and (2) computationally difficult to compute accurately on classical computers due to its complexity. Since exhaustive search or direct evaluation of θ is infeasible for large systems, we train `QE predictor`, a regressor, on small, classically simulable systems to estimate energy as the GA's fitness function. Consequently, the model approximates the ground state energy, aiming to minimise prediction error trends rather than exact values (see Equation 1). GA was chosen for its ability to handle mixed-type inputs, explore large search spaces, and, in this work, integrate code-level feasibility tests for hyperparameter assignments directly into the evaluation loop (*i.e.* the QE's test set). XGBoost was selected for its robustness on sparse, nonlinear, and floating-point data. XGBoost alone is insufficient, as it struggles on mixed-type inputs and could naively increase parameters such as `sampling_shots` or `iteration_max` to improve predicted performance without considering resource constraints. Importantly, our optimisation targets the QE implementation's hyperparameters, not the ML model's, distinguishing our work from typical ML hyperparameter tuning. Exploration of alternative regressors, ML hyperparameter tuning, or other modelling setups for `QE predictor` is left for future work.

4.1 Fitness Function

`AccelerQ` employs a fitness function, $f_{\text{QE}}(\boldsymbol{\theta}; \hat{H})$, defined in Equation 1 to iteratively optimise the hyperparameters of a given QE implementation, minimising error in the predicted ground-state energy of a given Hamiltonian $\hat{H}$. The QE hyperparameters, θ, is an ordered tuple represented as a vector of mixed types (float, boolean, integers). In the preparation stage, we define θ's structure per QE. ADAPT-QSCI and QCELS's are listed in §6. Example 1 shows possible differences between the default hyperparameters, $\boldsymbol{\theta}_{DEF}$, which were in the original QE implementation, the two variants of AccelerQ, one using only ML θ_{OPT}, and one combining ML with a GA guided by a test set $\boldsymbol{\theta}_{TEST}$.

Compressed Flattened Hamiltonian: In our setting, $\hat{H}$ is the *problem Hamiltonian*, encoding the total energy of the quantum system. It is represented in code as a `FermionOperator` object (from `OpenFermion`), which stores a set of term-coefficient pairs. Terms are floats, and coefficients are complex numbers. $\hat{H}$ is converted into a fixed-length numeric feature vector via the following steps:

(1) **Term extraction**: Retrieve all operator–coefficient pairs from the `FermionOperator`.
(2) **Flattening**: Recursively unpack any nested list or tuple structure into a one-dimensional list of floating-point numbers, discarding any imaginary component.
(3) **Filtering**: Remove small-magnitude values below a given threshold (*e.g.* $|x| \leq 0.2$).
(4) **Sorting**: Sort coefficients by absolute value to prioritise larger contributions.

Effect of Hyperparameter Optimisation for ADAPT-QSCI

Example 1. *For ADAPT-QSCI, the default hyperparameters (θ_{DEF}) are:*

100, 0.001, 0, 100, 100000, 1.00E−06, 5, 128, 0,<Compressed Flattened Hamiltonian>

Using these, the predicted lowest eigenvalue for a 20-qubit system is -21.102120951*, compared to the true value of* -22.046059902*. AccelerQ optimises the hyperparameters and may yield the assignment (θ_{OPT}):*

985, 7.64647e−03, 0, 344658, 49458, 5.67168e−05, 3, 77, 1,<Compressed Flattened Hamiltonian>

This results in a prediction of -21.513448108*, cutting the error by half. Using the hyperparameter validation set (θ_{TEST}), AccelerQ selects more reasonable values—for example, reducing the algorithm's maximum iterations from 344,658 to a much lower limit. The values in this example follow the order and format of the hyperparameters listed in Table 1.*

This produces a compact representation of $\hat{H}$ that captures both the structure of the interactions and their relative strengths, enabling regression models to correlate Hamiltonian characteristics with effective hyperparameter settings. The compressed, flattened Hamiltonian remains fixed during optimisation and is not mutated.

4.2 Hyperparameter Validation Test Set

Hyperparameter optimisation is inherently greedy: selecting the set with the minimum predicted value often yields configurations with a combination of a high number of shots and many iterations. While this may improve accuracy, it also leads to execution times spanning days or weeks. In simulations, running such setups locally may be inconvenient but manageable. However, the computational cost can become extremely expensive when deployed on quantum hardware. Moreover, many platforms cap the number of shots per execution or over time, further limiting the feasibility of high-resource configurations.

A QE's Test Set: We implement static and semi-dynamic validation tests to ensure sensible hyperparameter combinations. *Static tests* check constraints without execution of the QE implementation code, while *semi-dynamic tests* run selected functions without executing the full ADAPT-QSCI or QCELS pipeline. The tests aim to enforce some constraints roughly to avoid disrupting the overall optimisation process (*e.g.* limiting max iterations to 1000 when typically only 100 fit in ADAPT-QSCI). These tests are encoded into the *generator* with static tests running first, followed by more complex ones. Each QE implementation has its own test set, with some overlap. We list the test set for each QE implementation in §6.

4.3 Data Augmentation

ML algorithms operate in two primary phases: training and prediction. In a supervised learning context, the training phase requires a sufficiently large and representative dataset consisting of input instances and corresponding ground truth outputs. When this dataset is diverse and of high quality, the trained `QE predictor` model can generalise and make reliable predictions on unseen data. Therefore, the effectiveness of our approach strongly depends on the quality and size of the training dataset. However, obtaining a large and robust dataset for quantum simulations is challenging, primarily due to the high computational cost of accurately computing energy levels for large Hamiltonians on quantum hardware[6].

To address this challenge, we employ a domain-specific form of *data augmentation* (Ⓐ, Ⓑ, Ⓒ Figure 5). We generate multiple variants of input Hamiltonians by modifying their parameters

[6]For example, IBM's quantum computing is priced at $48 per minute https://www.ibm.com/quantum/pricing

or structures in physically meaningful ways that preserve essential characteristics. This includes truncating Hamiltonian terms below a meaningful threshold or adjusting coefficient cutoffs. These variations simulate plausible alternative quantum systems, effectively enriching the training set while preserving the statistical properties needed for generalisation. Since these transformations are applied to Hamiltonians with up to 16 qubits, we can compute accurate energy levels using classical state vector simulators, avoiding the need for costly quantum hardware. Moreover, because our systems are relatively small, the data augmentation process can be executed efficiently on standard classical hardware (*e.g.* x86 CPUs), without requiring GPUs or specialised accelerators.

Each training instance is $(\mathbf{x}_i, y_i)$, with $\mathbf{x}_i = (\boldsymbol{\theta}_i; \hat{H}_i)$ and $y_i = f_{QE}(\boldsymbol{\theta}_i; \hat{H}_i)$, where $\hat{H}_i$, is the compressed Hamiltonian (preserving essential physical properties while saving memory and avoiding over-fitting), f_{QE}, is the exact energy level (computed classically), and $\boldsymbol{\theta}$, the associated hyperparameters. To generate many $\mathbf{x}_i$, we augment by drawing $\hat{H}$ from open-source/academic benchmarks, and $\boldsymbol{\theta}$ is sampled at random within predefined type/range constraints.

We perform augmentation separately for each QE implementation because (1) hyperparameters differ across implementations, and (2) each implementation exhibits different behaviour, even when using a classical state vector simulator. We use the resulting dataset, (X, Y), to train a `QE predictor` model to predict suitable hyperparameters for unseen Hamiltonians with up to 28 qubits.

4.4 The QE Predictor Training

While computing the fitness function (or lowest eigenvalues) for QE with 20-30 qubits is feasible, doing so directly is computationally extremely expensive. Instead, we use a `QE predictor` model to approximate the relationship between hyperparameter choices. We collect data arrays from Hamiltonian systems with ≤ 16 qubits, of hundreds of computationally inexpensive samples (§4.3). These smaller systems typically have higher minimum energy levels. Nonetheless, the aim is not to compute the exact minimum eigenvalue but to identify hyperparameters likely to yield lower values. This reduces computational cost while guiding optimisation effectively, enabling evaluation of many assignments of $\boldsymbol{\theta}$ in §4.5 to make informed predictions for larger, more computationally demanding systems.

`AccelerQ` trains a `QE predictor` model in two phases (Ⓓ, Figure 5).

In the *data preparation phase*, we construct the feature array using data collected via data augmentation (§4.3), creating an (X, Y) augmented dataset; Y's values represent the true values obtained during the data preparation phase. `AccelerQ` pads all $x_i \in X$ to a fixed-size feature array of a predefined maximal size of the Hamiltonian systems. We include the compressed, flattened Hamiltonian (§4.1) in the feature array to establish a connection between the hyperparameter values and the specific characteristics of a Hamiltonian. These include *e.g.* dominant terms. The dataset is stored in a persistent storage for the training phase. In the *training phase*, `AccelerQ` trains a Regularising Gradient Boosting Regression (XGBoost regressor) using (X, Y) padded augmented dataset, for a given QE's hyperparameter vector and compressed Hamiltonians. The above can be summarised in the following steps:

Inputs: (X, Y), $\hat{H}$.

(1) Pads all vectors x_i in X to a fixed size: $|\theta_i| + max\{|\hat{H}|, \max_{x_i=(\boldsymbol{\theta}_i;\hat{H}_i)\in X} |\hat{H}_i|\}$.
(2) Splits into *training set* and *test set*.
(3) Trains XGBoost regressor with *training set*.
(4) Evaluates on the *test set*.
(5) Saves the trained model for use in §4.5.

Output: $\texttt{model\_file}_{QE\_IMPL\_NAME}$.

The `QE predictor` is generic, but training data depend on the QE implementation, solver, and hyperparameter generator. Trained models are saved as $\texttt{model\_file}_{QE\_IMPL\_NAME}$.

4.5 Optimisation

Once trained, `QE predictor` predicts optimal hyperparameters for larger-qubit Hamiltonians (Ⓔ, Figure 5). We use a `QE predictor` to evaluate the fitness function, f_{QE} (Equation 1) for the search-based optimisation process and apply mutations that adhere to the format of its hyperparameters.

In the *hyperparameter optimisation phase*, given a QE implementation (with its *generator* and `QE predictor`) and an input Hamiltonian $\hat{H}$ with more than 16 qubits–preprocessed (compressed and flattened) as in training–, `AccelerQ` initialises a population of candidate vectors $\{(\boldsymbol{\theta}_i; \hat{H})\}$, where each seed i is a hyperparameter assignment sampled by the *generator*. The fitness function, f_{QE}, evaluates each candidate by approximating the lowest eigenvalue for $(\boldsymbol{\theta}_i; \hat{H})$ configuration, utilising the `QE predictor` to compute an approximation of $f_{QE}(\boldsymbol{\theta}_i; \hat{H})$, and then selects the the best-performing candidates (those with the lowest predicted scores) for mutation in the next generation to create new seeds. A crossover operator then combines pairs of these vectors to generate new hyperparameter candidates, using methods such as averaging, selecting extreme values, and random values (see in [10]). `AccelerQ` occasionally introduces noise to diversify solutions and avoid local minima. The process continues until a stopping condition is met, such as reaching a maximal number of iterations. The vector $\boldsymbol{\theta}$ with the lowest error (or just the lowest predicted eigenvalue if the true value is unknown) is then returned as the optimal hyperparameter set. The procedure steps are as follows:

Inputs: `model_file`, `regressor`, `generator`, `test_static`, `test_semi_dynamic`, $\hat{H}$, $C = \emptyset$.

(1) **Load `QE predictor`:** $f_{QE} \leftarrow$ `load_model(model_file, regressor)`.

(2) **Seed Initial population:** Repeat $i = 1..500$:

- 2.a. Sample $\theta_i \leftarrow$`generator(`i`, opt_n_qubit)`
- 2.b. if fails `test_static` || `test_semi_dynamic`: discard θ_i else: $C \leftarrow C \cup \{(\boldsymbol{\theta}_i, f_{QE}(\boldsymbol{\theta}_i; \hat{H}))\}$

(3) **Iterative improvement:**

- 3.a. *Selection:* keep best ∼10%, lowest scores in C.
- 3.b. *Mutation:* repeatedly combine two, $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}_j$, thus $(\boldsymbol{\theta}_j, score_j), (\boldsymbol{\theta}_k, score_k) \in C$ using one of the crossover operators, resulting in a new mutant $\boldsymbol{\theta}_i$.
- 3.c. if fails `test_static` || `test_semi_dynamic`: discard θ_i else: $C \leftarrow C \cup \{(\boldsymbol{\theta}_i, f_{QE}(\boldsymbol{\theta}_i; \hat{H}))\}$
- 3.d. *Periodic pruning & noise:* every 5 rounds, prune the population (keep top ∼50%) and inject new random seeds via `generator` (filtered by tests) to maintain diversity.

Output: Return best $\theta^\star$ that is the x of $(x, y) \in C$ such that y is the smallest in C.

The `model_file` is unique per QE implementation (*i.e.* $\texttt{model\_file}_{QE\_IMPL\_NAME}$) and regressor is XGBoost (§4.4). `test_static` and `test_semi_dynamic` are the QE's test set (§4.2), require access to the QE implementation's code, and are applied in order of cost, with static tests run first. The set C, the seed set, contains pairs of hyperparameters and their corresponding fitness function scores. In our experiments, Step (3) ran for 50 iterations. We execute the QE implementation with $\boldsymbol{\theta}^\star$ (Ⓕ, Figure 5). We repeat the optimisation for each Hamiltonian system and for both ADAPT-QSCI and QCELS implementations in §7.

5 Implementation Details

We implemented `AccelerQ` (all stages in §4) in Python 3.10.12, using a GPU for training and keeping `QE predictor` models small for CPU deployment. We employ the open-source Python XGBoost library (eXtreme Gradient Boosting) [97]. We used the Python library *QURI Parts*[7] for the simulations while applying some resource consumption constraints [77]: a practical time limit of 6×10^5 seconds and a shot limit of 10^7 to mirror the limitation of the high cost of quantum computer execution. We used ADAPT-QSCI from [63] and QCELS from [18].

Each QE can run in a *classical mode* (an exact classical simulator) or an *evaluation mode* (an approximate quantum simulator). We restricted *classical mode* runs to a maximum of 16 qubits due to high computational complexity. *Evaluation mode* uses matrix product states (MPS) [6, 67, 105, 106] to store the system state more efficiently, enabling time- and memory-efficient simulation of larger systems, but at the cost of some approximation error.

A QE takes **number_qubits**, the flag **is_classical** (set to `True` during data collection and `True` or `False` otherwise), and $\boldsymbol{\theta}$ as defined in §2. It outputs the system's lowest energy prediction as a Python float (17-digit precision). Each QE uses fixed default hyperparameters ($\boldsymbol{\theta}$ vector), which differ between implementations and include the compressed, flattened Hamiltonian. $\boldsymbol{\theta}$ vectors must have a consistent length for the Python XGBoost library. Before training, vectors are padded to a predefined maximum length, split into training and testing sets, and then trained, tested, and evaluated. `AccelerQ` takes a QE implementation with its required inputs.

`AccelerQ` leverages Python's ability to pass functions as arguments, allowing it to accept QE implementations and their hyperparameter generators as inputs. The data augmentation, model training, and optimisation steps are implemented in Python and can work with any such QE implementation, although a wrapper and manual definition of constants as a hyperparameter problem are currently required. This modularity enables straightforward extension to other QEs, including those with hardware backends or simulator interfaces in C/Java, which we identify as promising future work.

6 Evaluation

We evaluated `AccelerQ`'s ability to further optimise ADAPT-QSCI and QCELS implementations, given a QE implementation and a system-specific Hamiltonian as input.

6.1 Methodology

Configurations. We consider three configurations in our evaluation of `AccelerQ`:

(1) **Baseline.** Executes QE implementations with fixed default hyperparameters.
(2) **ML-Only.** `AccelerQ` operates only with the `QE predictor`, excluding §4.2 and §4.5.
(3) **Full AccelerQ.** The complete approach as described in §4.

`DEFAULT` (Configuration 1) executes the QE implementations with their default hyperparameters, which are fixed across all systems and are as in [18, 63]. `OPT-ML-Only` (Configuration 2) excludes the use of genetic algorithms and hyperparameter tests and serves as a weaker variant of `AccelerQ`: technically, we generated random seeds for the same number of iterations and selected the one with the minimum score, no tests and no mutations.

[7] https://pypi.org/project/quri-parts/

Table 1. Parametrisation of ADAPT-QSCI and QCELS hyperparameters. # is the number of items.

Parameter	Type	Range	Default	Description
ADAPT-QSCI Hyperparameters				
num_pickup	`int`	[50, 10^3]	**100**	Controls #terms retained in the compressed Hamiltonian.
coeff_cutoff	`float`	[1e-8, 1e-2]	**0.001**	Complimenting num_pickup: Hamiltonian terms with coefficients below it are excluded.
self_selection	`bool`	True, False	**False**	If True, forces working in subspace.
iter_max	`int`	[10, 10^5]	**100**	Maximum iterations for the algorithm.
sampling_shots	`int`	[10, 10^6]	10^5	#sampling shots for measurements per iteration.
atol	`float`	[1e-8, 1e-4]	**1e-6**	Absolute tolerance for convergence criteria.
final_sampling_shots_coeff	`int`	[1, 9]	**5**	How many more shots to use in the calculation if the same operator appears twice or operator parameter is close to 0.
num_precise_gradient	`int`	[35, 300]	**128**	#operators from pool to calculate gradient more precisely.
reset_ignored_inx_mode	`int`	[0, 100]	**0**	#iterations to pass before reusing an operator in ansatz.
QCELS Specific Hyperparameters				
ham_terms	`int`	[50, 10^3]	**200**	#terms retained in the Hamiltonian after truncation.
ham_cutoff	`float`	[1e-8, 1e-3]	**1e-9**	Same as **coeff_cutoff** in ADAPT-QSCI.
delta_t	`float`	[1e-3, 0.3]	**0.03**	Time step for the simulation or evolution of the system.
n_Z	`int`	[5, 25]	**25**	#points used in fitting the time evolution.
alpha	`float`	[0.5, 1]	**0.8**	Scalar to control parameters' weight in Equation 2.

Experimental Procedure. We trained two `QE predictor` models—one per QE implementation—on data from classically simulable systems up to 16 qubits. These models were then deployed to optimise 16 larger Hamiltonians of 20, 24, and 28 qubits with known lowest eigenvalues, using the ADAPT-QSCI and QCELS implementations (§3). The process used a QE implementation, its trained model, and a system (Hamiltonian) and returned optimal hyperparameters. We then assessed the performance with Configurations 1-3 and addressed *RQ*s 1-4 stated in the Introduction (§1).

6.2 Experimental Setup

We describe below the experimental setup for the preparation of QE implementations and Ⓐ, Ⓑ, Ⓒ and Ⓓ in Figure 5 in §4.

Source of Hamiltonians. We use two sources of Hamiltonians: **(1) QunaSys's datasets**[8] [75, 77], and **(2) commonly used open-source molecular Hamiltonians**, including H2O, LiH, BeH2, Hemocyanin [3], and Hydrogen chain. Evaluation was performed on 20-, 24-, and 28-qubit Hamiltonians using predictions trained on smaller systems. Due to the high cost of quantum hardware and the long runtime of QEs on 20+ qubits in the NISQ QC, all testing was done in simulation.

Data Augmentation. We extracted 66 files in a classical mode. ADAPT-QSCI produced 4,760 records (757 MB), while QCELS, being more efficient on smaller systems, yielded 14,510 records (4868 MB), including 400 and 5,550 records respectively from source (1) Hamiltonians and the rest from source (2). For 4- 6-, 7-, 8-, 10-, 12-, 14- and 16-qubit systems, we utilised 60, 750, 500, 500, 1500, 450, 800 and 200 records for ADAPT-QSCI hyperparameters, and 60, 750, 500, 500, 1000, 6600, 2100 and 3000 for QCELS hyperparameters, respectively.

Parameterisation. The ADAPT-QSCI and QCELS implementations [18, 63] include default hyperparameters controlling their operation, used as a baseline for comparison in our evaluation (*i.e.* `DEFAULT`). We summarised these in Table 1. All parameters of both implementations follow a uniform distribution, except `iter_max` and `sampling_shots`, which follow a custom multi-tiered

[8] *QunaSys's Hamiltonians*: 4 and 12 qubits for seeds __00 to __04, and of 20 and 28 qubits for seeds __00 to __04.

Table 2. Hyperparams. Tests of ADAPT-QSCI & QCELS Impl.; S: Static tests; SD: Semi-dynamic tests.

#Test	Impl.	Type	Relevant Hyperparameters	Description of the Test
1	ADAPT-QSCI	S	iter_max, sampling_shot	No overshooting or using >1000 iterations.
2	Both	SD	num_pickup, coeff_cutoff, or ham_terms, ham_cutoff	Compressed Hamiltonian size is reduced meaningfully.
3	Both	SD	same as #2	Checks if cut-off value is effective in reducing terms: this is similar to #2, but set ham_terms to be the maximal (*i.e.* #terms in the original system).
4	Both	SD	num_pickup or ham_terms	num_pickup is reasonable based on Hamiltonian size.
5	QCELS	S	n_z	n_z is in [5,30].
6	ADAPT-QSCI	S	self_selection, system type	Checks self_selection is sensibly set based on Fermionic and particle-conserving properties.
7	ADAPT-QSCI	S	self_selection, iter_max, reset_ignored_inx_mode	If self-selection is enabled, max iterations exceed reset iterations.
8	QCELS	S	delta_t	delta_t is within a reasonable range.
9	QCELS	S	alpha	alpha is in [0.5,0.9] for stability.

distribution that uniformly at random picks 10^i, and then uniformly samples an Integer in $[10^i, 10^{i+1}]$. We set the hyperparameter ranges to fit the physical problem context. For `QE predictor`, we defined $x \in X$ vector as the compressed Hamiltonian (by removing terms with absolute coefficients below 0.05) and its hyperparameters, normalised to the size of 28-qubit systems. The $y \in Y$ vector is the predicted lowest eigenvalue, yet the ys values are relative approximations (*i.e.* the ys values are approximations that capture relative relationships rather than absolute meaningful values).

Default Values. Default values (Table 1, Default column, `DEFAULT` Configuration) were overridden when using `AccelerQ` suggestions tailored per Hamiltonian system using values drawn from the range (Range column, Configurations `OPT-ML-Only` and `FULL-AccelerQ`).

Tests. We wrote a set of tests for ADAPT-QSCI and QCELS summarised in Table 2, including the implementation and the hyperparameters the test is relevant to (Impl. and Relevant Hyperparameters columns), the type of the tests (static or semi-dynamic test; Type column), and the test description (Description of the Test column).

Model Extraction. `QE predictor` models' sizes were 1.1 MB for ADAPT-QSCI and 2.86 MB for QCELS, trained with data extracted on an exact classical simulator from up to 16-qubit systems.

Machine Setup. Models were trained using XGBoost (XGBRegressor, v2.1.1) on a single GPU core, NVIDIA 12GB PCI P100 GPU, 12 GB VRAM, running Ubuntu 22.04.4 LTS [26]. Simulations and model deployment ran on a virtual machine with Ubuntu 20.04.2 LTS (x86_64), hosted on a single-socket AMD EPYC 7313P CPU (3.0 GHz, 16 cores, 2 threads/core). Training data was collected and processed entirely on a CPU.

7 Results

We now evaluate `AccelerQ`'s ability in optimising hyperparameters of QE implementations to improve accuracy and efficiency across 16 Hamiltonian systems. We analysed results for `QE predictor` deployment to generate new hyperparameters suggestions (Ⓔ, Figure 5) in §7.1, and execution of QE implementations with these hyperparameters (Ⓕ, Figure 5) in §7.2. Model deployment ran on an exact classical simulator, while QEs execution with optimised hyperparameters used an MPS-based quantum simulator (§5). Full data and tables are available in our artifact §12.

7.1 RQ_1: Model Deployment

We deployed the two models on 16 Hamiltonian systems (§6.2). Figure 6 (ADAPT-QSCI) and Figure 7 (QCELS) present the hyperparameter values under three configurations: `DEFAULT` (DEF), `OPT-ML-Only` (OPT), and `FULL-AccelerQ` (TEST). The x-axis labels these configurations for each hyperparameter. The default hyperparameters (DEF) are listed in Table 1, while OPT and TEST values were obtained using `OPT-ML-Only` and `FULL-AccelerQ` configurations, respectively. Columns A–I correspond to the predicted optimal hyperparameters for ADAPT-QSCI, and Columns A–E for

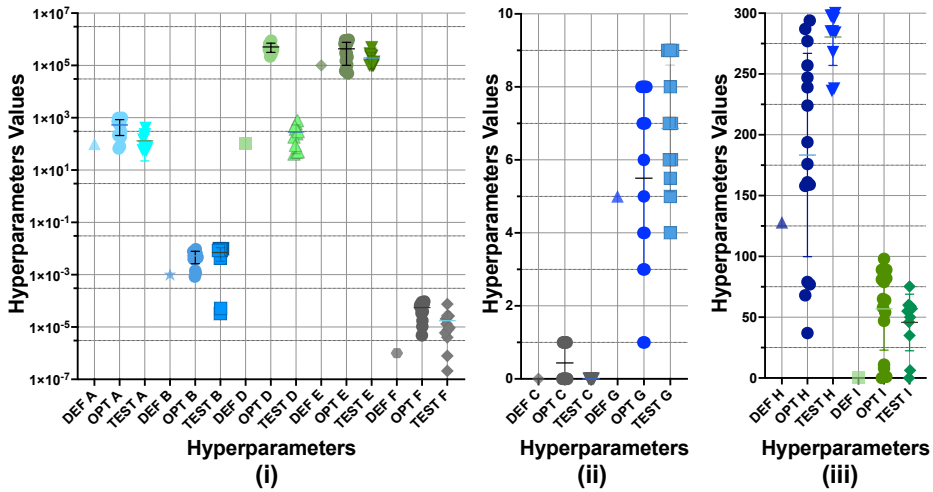


Fig. 6. **ADAPT-QSCI's Hyperparameters: Default** (DEF) **vs predicted values.** (i)'s Y-axis in log scale, $\log_{10}$. X-axis labels: **A**: num_pickup, **B**: coeff_cutoff, **C**: self_selection, **D**: iter_max, **E**: sampling_shots, **F**: atol, **G**: final_sampling_shots_coeff, **H**: num_precise_gradient & **I**: reset_ignored_inx_mode.

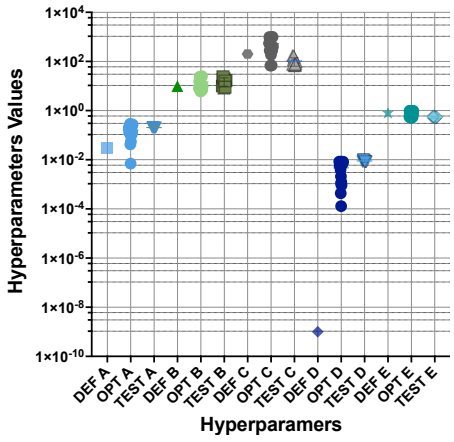


Fig. 7. **QCELS's Hyperparameters: Default** (DEF) **vs predicted values.** Y-axis in log scale, $\log_{10}$. X-axis labels: **A**: delta_t, **B**: n_Z, **C**: ham_terms, **D**: ham_cutoff and **E**: alpha.

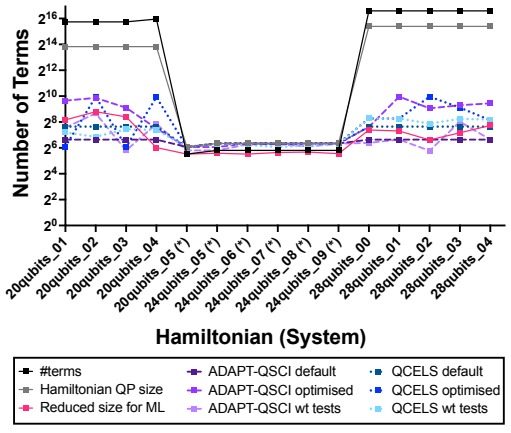


Fig. 8. **Number of Hamiltonian terms** (Y-axis, $\log_2$ scale) for `QE predictor` queries and QE implementation executions across different systems (X-axis), shown as raw and compressed sizes.

QCELS. Note that the value presented in Figure 6 for the predicted `iter_max` was capped during execution[9] to ensure that we do not have unlimited resources.

Figure 6 shows that OPT and TEST generally select higher values than DEF, except for stopping-related parameters (`final_sampling_shots_coeff` and `num_precise_gradient`), which remain closer to DEF. OPT typically reduced per-iteration precision, lowering `sampling_shots` while increasing `atol`, `coeff_cutoff`, and `iter_max`, leading to more total iterations. When both `iter_max` and `sampling_shots` were pushed to maximum values, ignoring their correlation, execution was capped, resulting in fewer iterations in practice. TEST did not exhibit this behaviour: it generally increased shots per iteration while keeping iteration counts moderate. Except that, TEST commonly aligned around the same range of values as OPT but aligned more with DEF on `self_selection` and `iter_max`, likely due to the constraints imposed by the hyperparameters tests, and consistently favoured higher `final_sampling_shots_coeff`, improving prediction quality.

Figure 7 shows that OPT tended to select higher values for `n_z` and `delta_t`, with no clear preference for alpha and favoured increasing the number of terms retained in the Hamiltonian (except for 20qubits_01 and 20qubits_03). This may seem to be contradicted by also favouring larger `ham_cutoff` values to decrease the number of terms retained in the Hamiltonian. Yet, for systems at this scale, the `ham_cutoff` value was always small enough to remove no additional terms beyond those excluded by `ham_terms`. TEST was generally aligned with OPT but utilised a narrower value range, often selecting from the higher end. Exceptions were `ham_terms` and `alpha`, where TEST chose values lower than OPT and DEF.

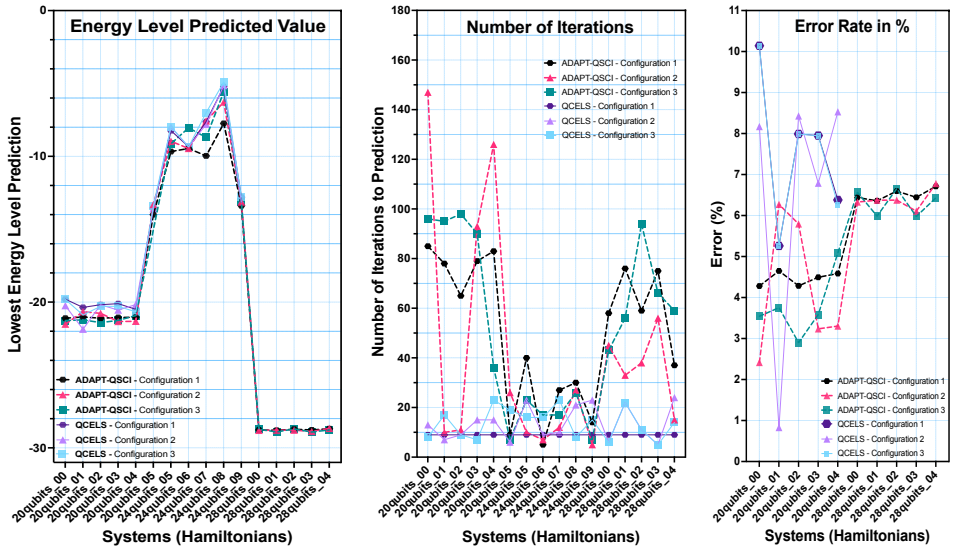
Hamiltonian Systems Size. The QE implementations truncate Hamiltonians based on hyperparameters such as coefficient cutoff and number of terms (Table 1). Thus, the choices in Figure 6 and Figure 7 under DEF, OPT, and TEST affected Hamiltonian size, which vary in their initial size and complexity (§6.2). We examined this effect to reveal structural differences and assess the relative difficulty of problem instances.

Figure 8 shows the individual number of terms in each Hamiltonian system in `FermionOperator`[10] format ("#terms" line) and the number of terms utilised in the `QE predictor` models during data augmentation, training and deployment of the models ("Reduced size for ML" line). The remaining lines capture sizes after the Jordan–Wigner transformation [65], including both the untruncated Hamiltonian ("Hamiltonian QP size") and its truncated form under the three hyperparameter configurations (DEF, OPT, TEST) of ADAPT-QSCI and QCELS. The ML pipeline used different cutoffs and compression than the Jordan–Wigner ones. Open-source molecular Hamiltonians are marked with an asterisk. Across all Hamiltonian representations in Figure 8, the open-source molecular Hamiltonians contain far fewer terms (≤ 46) than the QunaSys Hamiltonians ($\geq 54k$). Even after compression, the molecular Hamiltonians remained much smaller, indicating structural differences between the sets.

RQ_1 **Answer.** We observed clear shifts in hyperparameter values. In ADAPT-QSCI, OPT and TEST typically increased `coeff_cutoff` and `iter_max`, but TEST increased `sampling_shots` to kept `iter_max` moderate. In QCELS, OPT and TEST raised `ham_cutoff` but aimed to retain more Hamiltonian terms similar to DEF. These effects were pronounced on large QunaSys Hamiltonians ($\geq 54k$ terms) than on smaller open-source ones (≤ 46 terms).

[9]The platform automatically stopped the computation once the maximum number of shots, 10 000 000, was reached, which is iter_max=1e7/sampling_shots; same limitation of the maximum number of shots applied to QCELS and is common in quantum platforms.

[10]the chemistry form, https://github.com/quantumlib/OpenFermion/tree/master



(i) (ii) (iii)

Fig. 9. Comparison of ADAPT-QSCI and QCELS, with 3 different configurations for each.

7.2 RQ_2–RQ_5: Execution with Different Hyperparameters

We ran six experiments in total, using the DEF, OPT, and TEST values of hyperparameters from RQ_1 for ADAPT-QSCI (Figure 6) and QCELS (Figure 7), as follows:

- *ADAPT-QSCI – Configuration 1.* ADAPT-QSCI with default hyperparameters (`DEFAULT`).
- *ADAPT-QSCI – Configuration 2.* ADAPT-QSCI with `OPT-ML-Only` predicted hyperparameters.
- *ADAPT-QSCI – Configuration 3.* ADAPT-QSCI with `AccelerQ` as in §4 (`FULL-AccelerQ`).
- *QCELS – Configuration 1.* QCELS with default hyperparameters (`DEFAULT`).
- *QCELS – Configuration 2.* QCELS with `OPT-ML-Only` predicted hyperparameters.
- *QCELS – Configuration 3.* QCELS with `AccelerQ` as in §4 (`FULL-AccelerQ`).

Each experiment was repeated 10 times to account for variance, with the best-performing (lowest valid) result reported. `FULL-AccelerQ` was executed only twice per Hamiltonian system—far fewer opportunities for improvement—placing it at a disadvantage compared to other configurations. This limitation was necessary due to the high computational cost: evaluating a single Hamiltonian with 10 repetitions can take $\approx$ 2 months on CPU[11]. Overall, these settings (2 QE implementations $\times$ 3 configurations, repeated 10 or 2 times) yielded $16 \times 2 \times 2 \times 10 + 16 \times 1 \times 2 \times 2 = 704$ executions, enabling a systematic comparison across Hamiltonians, configurations, and algorithmic behaviours.

Figure 9 summarises the results of executing ADAPT-QSCI and QCELS on a quantum simulator. We evaluated 16 Hamiltonians of 20-, 24-, and 28-qubit systems (x-axis). Results for 28-qubit QCELS are intentionally omitted, as they appeared to be likely invalid, discussed further in §8. Error rates in Figure 9(iii) are reported only for QunaSys Hamiltonians, since reference values for open-source molecular Hamiltonians were unavailable at the time of the experiments. At that stage, we operated with **`is_classical`**`:=False` (see §5). We evaluated three metrics:

- Figure 9 (i) reports the minimum energy level estimated;

[11]GPU runs could have reduced runtime substantially, but we had access only for a few days, insufficient for full experiments.

- Figure 9 (ii) presents the number of iterations required to reach estimation;
- Figure 9 (iii) shows the error rate in %, computed for Hamiltonians with known solutions [75, 77].

Figure 9 (i) shows that default parameters (DEFAULT) performed better for the open-source molecular Hamiltonian systems ($\leq$ 46 terms), while QunaSys's Hamiltonians ($\geq$ 54,000 terms) performed better with the AccelerQ optimisation (Configurations 2-3). QunaSys's datasets (10 systems): ADAPT-QSCI Configuration 2 achieved the best result in five Hamiltonian systems, followed by ADAPT-QSCI Configuration 3 in four Hamiltonian systems, and QCELS Configuration 2 in one Hamiltonian system. No other configurations, including the defaults, achieved top performance. Dataset of commonly used open-source molecular Hamiltonians (6 systems): ADAPT-QSCI Configurations 1 and 2 performed best for system 24qubits_06, ADAPT-QSCI Configuration 3 for 20qubits_05 and ADAPT-QSCI Configuration 1 for the rest of the four systems.

Figure 9 (ii) presents the number of iterations used by each QE implementation to approximate the lowest eigenvalue. All executions remained below 150 iterations, even when higher iteration counts were permitted by the hyperparameter settings (shown in Figure 6 and Figure 7). In general, QCELS required fewer iterations on average, 9.0 for Configuration 1 (DEFAULT), 13.69 for Configuration 2 (OPT-ML-Only), and 13.69 for Configuration 3 (FULL-AccelerQ), compared to ADAPT-QSCI, which averaged 51.25, 41.31, and 50.37, respectively. While this reflects the fundamentally different nature of the two algorithms, FULL-AccelerQ commonly required slightly more iterations than OPT-ML-Only.

Figure 9 (iii) presents the error rate for QunaSys's datasets (10 systems). For 20-qubit systems, ADAPT-QSCI Configuration 3 (FULL-AccelerQ) achieved the lowest average error at 3.77%, followed by ADAPT-QSCI Configuration 2 (OPT-ML-Only) at 4.20%, and ADAPT-QSCI Configuration 1 (DEFAULT) at 4.46%. Whereas QCELS OPT-ML-Only reached 6.55%, while DEFAULT had 7.55%, and FULL-AccelerQ showed negligible improvement against the default, with an average error of 7.52%. For the 28-qubit systems, we excluded QCELS due to instability at this scale (see §8). ADAPT-QSCI FULL-AccelerQ achieved the lowest average error at 6.32%, followed by OPT-ML-Only at 6.39%, and DEFAULT at 6.51%.

Answer to RQ_2: Ability of Optimisation via ML alone to Accelerate and Improve QE Executions (Configuration 1 vs Configuration 2). Results indicate that Configuration 2 (OPT-ML-Only) provides limited improvements when generalising from small ($\leq$16 qubits) to larger (20–28 qubits) systems. While it outperformed its default counterpart (DEFAULT) on certain QunaSys Hamiltonian systems, the improvements in accuracy remained relatively modest. These findings suggest that while generalisation is possible, its effectiveness depends on the Hamiltonian's complexity and structure.

(QunaSys's datasets, 20-qubit) OPT-ML-Only exhibited greater error variability (0.83%–8.53%) compared to DEFAULT (4.28%–10.14%). While QCELS OPT-ML-Only achieved the lowest error (0.83%) within just seven iterations for 20qubits_01 system, ADAPT-QSCI OPT-ML-Only performed better in other cases. **(Open-source datasets, 20qubits_05 and 24-qubit)** ADAPT-QSCI DEFAULT outperformed all other configurations, suggesting that for Hamiltonians with relatively few terms, hyperparameter tuning does not enhance accuracy. **(QunaSys's datasets, 28-qubit)** ADAPT-QSCI OPT-ML-Only outperformed DEFAULT, often achieving error reductions of an order of magnitude. In contrast, DEFAULT showed only marginal advantages in cases where they outperformed OPT-ML-Only (*e.g.* 0.x vs. 0.0x error differences), suggesting that hyperparameter optimisation can lead to substantial accuracy gains, even if not universally superior to defaults.

> RQ_2 **Answer.** OPT-ML-Only showed limited overall gains, performing better primarily with Hamiltonians containing hundreds of terms. Likely, a more refined model incorporating Hamiltonian characteristics may be necessary for assessing the impact of optimisation.

Answer to RQ_3: Assessing additional efforts beyond ML (Configuration 2 vs Configuration 3). Our evaluation shows that Configuration 3 (`FULL-AccelerQ`) generally outperforms Configuration 2 (`OPT-ML-Only`) in terms of error rate, though it does not always achieve the best result. **ADAPT-QSCI:** `FULL-AccelerQ` achieved the lowest error rates for QunaSys's datasets: 3.77% (SD ±0.8) and 6.32% (SD ±0.32), for 20- and 28-qubit systems, respectively, surpassing `OPT-ML-Only`: 4.20% (SD ±1.71) and 6.39% (SD ±0.24). **QCELS:** `FULL-AccelerQ` did not yield any gains, possibly due to either hyperparameter values or hyperparameter tests not significantly affecting how QCELS internally operates; for 20-qubit systems, `FULL-AccelerQ` was 7.52% (SD ±1.87) while `OPT-ML-Only` was 6.55% (SD ±3.27). This was unexpected: we implemented QCELS ourselves and expected greater control over its behaviour, unlike ADAPT-QSCI, whose implementation details were less familiar to us. Yet, ADAPT-QSCI is designed for NISQ devices with tunable parameters that expose more learnable patterns, while QCELS is less sensitive to initialisation and more constrained by physics-based evolution, reducing the benefits of `FULL-AccelerQ`. For open-source datasets, results were inconclusive: `OPT-ML-Only` outperformed `FULL-AccelerQ` in some cases, and vice versa.

> RQ_3 **Answer.** `FULL-AccelerQ` showed more consistent performance, with its genetic algorithm and test filtering refining hyperparameter selection, with conclusive improvements across complex Hamiltonians with hundreds of terms. For open-source datasets of Hamiltonians with ≤ 46 terms, however, no consistent gains were observed with either `OPT-ML-Only` or `FULL-AccelerQ`.

Answer to RQ_4: Scalability of Configurations 1, 2 and 3. We compare the performance of all three configurations (`DEFAULT`, `OPT-ML-Only`, `FULL-AccelerQ`) to draw overall conclusions about their effectiveness and scalability. Scalability was assessed by measuring iteration counts and error rates (only for QunaSys datasets) of Hamiltonian systems of increasing size.

ADAPT-QSCI: `OPT-ML-Only` required fewer iterations on average than `FULL-AccelerQ` or `DEFAULT` (see Figure 9 (ii) discussion), with a general trend of using fewer iterations as the number of qubits increased from 20- to 28-qubits, indicating manageable scaling. We observed a reduction in error from 5.48% (SD ±1.09) of the defaults to 5.3% (SD ±1.63) with `OPT-ML-Only` and further to 5.05% (SD ±1.46) with `FULL-AccelerQ`. With respect to the system's size, error rates, however, rose from $\sim$ 2–5% to $\sim$ 6–7% (Figure 9 (iii)). **QCELS:** Iteration counts were low across configurations (Figure 9 (ii)) with no clear trend. For 20-qubit, `OPT-ML-Only` optimisation yielded limited error reduction, from 7.5% (SD ±1.85) to 6.5% (SD ±3.27), with no gain with `FULL-AccelerQ` for QCELS. In 28-qubit systems, `AccelerQ` failed to scale, as discussed in detail in §8.

> RQ_4 **Answer.** Compared to ADAPT-QSCI `DEFAULT`, `OPT-ML-Only` generally reduced the iteration count, whereas `FULL-AccelerQ` slightly increased it but delivered higher accuracy. By contrast, QCELS `OPT-ML-Only` and `FULL-AccelerQ` raised the iteration count by about 52% (from 9 to 13.69), though the values remain within a sensible range. `DEFAULT` configurations showed the lowest variance, offering stability at the expense of accuracy, while both `OPT-ML-Only` and `FULL-AccelerQ` reduced error rates, indicating the potential of optimisation.

Answer to RQ_5. The results show that variance in the final energy in QE implementations across Hamiltonians tends to decrease for QunaSys's dataset and to increase for the open-source datasets as system size increases. A similar trend was observed in error rates. Overall comparisons of the three configurations (`DEFAULT`, `OPT-ML-Only`, `FULL-AccelerQ`) were presented earlier (Figure 9). Here, we analyse the results per QE separately to highlight additional trends. **ADAPT-QSCI**: Across the benchmarked Hamiltonians, the three configurations achieved 5, 6, and 6 wins for `DEFAULT`, `OPT-ML-Only`, and `FULL-AccelerQ`, respectively. This indicates that while `OPT-ML-Only`

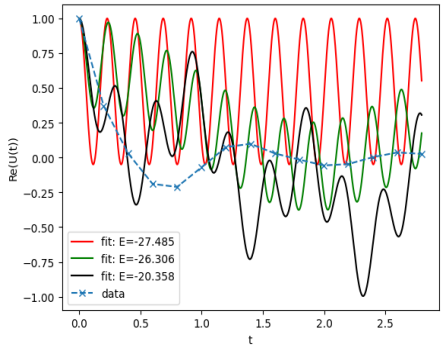


Fig. 10. The original high-frequency fit uses a period of approximately the spacing between the data. Damped oscillations were observed in the simulated quantum algorithm's output.

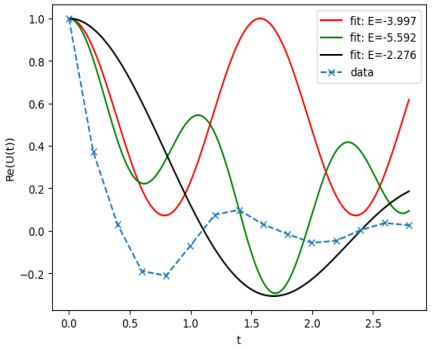


Fig. 11. An updated fitting procedure which prevents high frequency fits, but does not significantly improve the results in this case because of the error in the collected time evolution data.

and `FULL-AccelerQ` often outperformed the defaults, no single configuration dominated consistently across all systems. Instead, the advantage appears to depend on the Hamiltonian characteristics, with `FULL-AccelerQ` showing slightly stronger error rates than `OPT-ML-Only`, as discussed in RQ_3. **QCELS, 20- & 28-qubits**: The corresponding win counts were 2, 7, and 2 for `DEFAULT`, `OPT-ML-Only`, and `FULL-AccelerQ`. Here, `OPT-ML-Only` clearly outperformed the other two configurations, suggesting that unconstrained hyperparameter optimisation had the strongest impact in this implementation (discussed already in RQ_3). QCELS failed to scale effectively to 28-qubit systems as discussed in §8.

Furthermore, several Hamiltonian systems (particularly from the QunaSys dataset) were sensitive to non-default hyperparameters, occasionally crashing the ADAPT-QSCI and QCELS implementations. This can impact the reproducibility of specific best results tied to particular hyperparameter values. Nevertheless, given the significant overall reduction in error rates, even slight variations in hyperparameters are still likely to improve outcomes over the default configuration. We expand on these issues in §9.

RQ_5 **Answer.** Overall, the results remain inconclusive, with observed differences driven more by the underlying QE implementation and Hamiltonian system complexity than by the optimisation.

8 Discussion

Scalability analysis of the optimisation procedure is hampered by the limitations of current quantum platform simulators. Beyond comparing the results to known reference values, we also examined the underlying fitting process to assess whether the observed outcomes were achieved for the right reasons, rather than by chance, as happened with 28-qubit systems from the QCELS results. We excluded these results and elaborated on the findings that led to this decision.

For larger system sizes, the time to evaluate a large circuit can quickly become prohibitive for running many algorithm iterations, even with access to large compute resources and sufficient memory to store the system's state. These challenges prevented the successful evaluation of the QCELS implementation at 28 qubits with a strong damping effect on the oscillations of time evolving expectation value. Possible causes of such an effect could include the Trotter error [83, 88] introduced by the implementation of the Hamiltonian evolution unitary or the truncation of entanglement

between qubits in the *Matrix Product State* (MPS) simulator [6, 67, 105, 106]. The results we obtained when using 28 qubits in the MPS simulator are shown in Figure 10, a decay of the oscillation can be observed in the data which results in an erroneous fit with a frequency close to that of the spacing between the data points. In Figure 11, we forbid a fit with a period smaller than the spacing of the data, however, the decay of the oscillations continues to prevent a good fit to the data. QCELS provides an elegant approach to solving a Hamiltonian by its direct encoding of time evolution, but its performance is highly sensitive to simulation fidelity and algorithmic discretisation, making it difficult to tune. In comparison, the ADAPT-QSCI implementation continues to work well up to 28 qubits, with its formulation as a classical solver acting in a subspace defined from the measurements on a quantum computer being well suited to working at these moderate system sizes. This approach also works well at mitigating the impact of inexact quantum evolution from noise on a quantum device or error in the classical simulation of the quantum algorithm.

9 Threats to Validity

Internal Validity. Our results are subject to several internal threats. First, we observed that QE implementations are sensitive to the Python environment. Minor differences in library versions could lead to misleading evaluation and results; *e.g.* with ADAPT-QSCI and program seed 24qubit_07: we observed that the total number of iterations was too low (*i.e.* #4). When reverting[12] the Python libraries this number became sensible (*i.e.* #21). Second, the QE implementations were not always able to support all hyperparameter values, resulting in execution failures; *e.g.* Python execution was out-of-resources with ADAPT-QSCI (24qubit_07) or a segmentation fault occurred with QCELS (20qubits_04) and ADAPT-QSCI (20qubits_03) in the quantum simulator[13]. We carefully documented and controlled these environments to mitigate such issues, including a Docker container and a requirement file. We reran failed cases to obtain the required number of successful runs in our evaluation. Lastly, we tested `AccelerQ` beyond the final output of the QE implementations because even when the predicted lowest eigenvalue appears low and within acceptable bounds, it may stem from the wrong reasons, as discussed in §8.

External Validity. Challenges in understanding quantum information and the limited availability of suitable datasets, even after data augmentation, make it difficult to predict when and how ML generalisation will succeed. Nonetheless, ML predictions can significantly optimise costly quantum executions and improve the accuracy of the results. Further, the generalisability of our findings is constrained by the use of a quantum simulator, which may not fully reflect real-world quantum hardware behaviour in terms of cost and noise. In our evaluation, we restricted quantum resources in simulation to mirror better quantum hardware (§5), but more sophisticated and realistic quantum simulators (general behaviour and noise) are required.

Transferability. `AccelerQ` is adaptable to different datasets, with the primary limitation being the Hamiltonian size. Our approach can be ported to other platforms, provided they are open-source[14] and, modulo some modifications. These, however, required appropriate input: 1) Adapting to other QE implementations requires a hyperparameters generator tailored to the target implementation, re-running the full pipeline of data augmentation and model training, which is also implementation-specific; and 2) Adapting to other problem domains requires a set of small Hamiltonian systems (typically less than 16 qubits) relevant to that domain, with classically computed solutions for training. Additionally, as discussed in the internal validity section, we observed that QE implementations are sensitive to Python package versions. Even minor version mismatches (*e.g.* in xgboost,

[12]We reverted `xgboost`, `scikit-learn`, `numpy`, `cirq` and `qiskit` to be `2.1.0`, `1.5.0`, `1.23.5`, `1.1.0` and `0.41.1`.
[13]This error came from a method call to `add_single_qubit_gate` in `quri_parts`.
[14]When selecting a quantum platform, consider this requirement and its compatibility and transferability to other platforms.

numpy, or qiskit) can lead to unoptimised behaviour. Therefore, pinning package versions and containerisation (*e.g.* via Docker) are required for reproducibility. More general recommendations related to reproducibility are summarised below.

ML holds significant potential for optimising quantum calculations but remains non-trivial. A key limitation we identified is the absence of unified quantum datasets tailored for training ML models; developing such datasets would greatly benefit future research and improve the transferability of ML-based approaches like `AccelerQ`.

Training, deploying and evaluating a model directly with the Hamiltonian representation of quantum systems appears promising. However, small variations in the quantum training data significantly impacted model performance. It is therefore crucial to validate datasets with expert knowledge. For instance, cut-off strategies and padding with zeros for Hamiltonian systems with fewer than 100 terms performed poorly due to sparsity (as in §7). Further, training on datasets with varying numbers of qubits substantially influenced results (like in §8). A deeper investigation into the properties required for validating quantum datasets would benefit industry and academia. Another key challenge is the limited availability of comprehensive noise data from quantum hardware, restricting the ability to transition from simulation to real hardware.

10 Related Work

Challenges in Quantum Computing. QC was conceptualised initially to simulate quantum mechanics using computers *"built of quantum mechanical elements which obey quantum mechanical law"* [31]. Later, it was found that QC could have several potential applications and offer significant speed-up over classical computing [7–9, 13, 23, 37]. In 1994, Shor's proposal of a polynomial-time algorithm for prime factorization and discrete logarithms on a quantum computer raised enormous interest due to its potential threat to modern RSA cryptosystems [81]. Soon after, Grover introduced a fast database search on quantum computers [37] that promised quadratic speed-up over the best classical algorithm. The resulting potential speed-up is often referred to as "quantum supremacy" [5]. Several studies apply SE techniques to optimise quantum computing [33, 35]. Noticeably, testing [60, 82], debugging [58, 68, 80], verification [52, 100] approaches, and efficient synthesis techniques [47, 69, 90] have been found to be beneficial in quantum software development [60, 68, 82, 100].

Demonstrating quantum supremacy on real hardware remains a long-standing challenge, especially at a scale where quantum devices would solve real-life calculations. Although quantum supremacy seems difficult to achieve soon, NISQ algorithms—Imperfect hardware is often called Noisy Intermediate-Scale Quantum (NISQ) devices—are a prominent example that hybrid systems combining small quantum circuits with classical computations could present some computational advantages, i.e., a quantum advantage [72]. Most agree this stage of QC will likely last for the next few years if not decades, and refer to it as the NISQ era [72]. Variational Quantum Algorithms (VQA) are the most common example of an efficient combination of a reduced quantum circuit inside a classical optimisation loop [87]. Other algorithms use classical optimisation to enhance quantum calculations, such as QCELS [24, 25] that uses a fitting procedure to extract information from quantum calculations. Because of their prominent role in modern Quantum Computing research and industrial applications, we chose to focus our study on Variational and QCELS Algorithms.

Machine Learning for Quantum Software Engineering. ML algorithms are increasingly used to improve and automate SE tasks [27, 39, 79, 92, 94, 95, 101, 104], especially after the advent of Large Language Models (LLM), with common applications in SE, including optimisation, code generation, bug detection and automated testing [11, 14, 21, 28, 45, 91]. Furthermore, connections between ML and QC have been broadly explored, both to optimise ML with QC and to optimise QC with ML [30, 70, 93]. Yet, the applications of ML in QC [78] remain a very new and open

field of research. While quantum machine learning (QML) offers potential speedups in ML tasks [89], very recent work also demonstrates promising results in applying ML to optimise quantum computations [16, 53]. Nevertheless, these methods are usually applied on the quantum circuit itself, rather than the quantum implementation (or the program), on a few qubits [53], and are not Hamiltonian-specific. Even approaches that leverage Hamiltonian information within variational eigensolvers [64] remain limited to QE-specific hyperparameter tuning. `AccelerQ` takes a different approach from common QE optimisation methods—that either optimise without incorporating the target Hamiltonian into the process or are restricted to fixed system sizes [16, 86, 102], or rely on manual code rewriting, or even complete reimplementation, in the hope of improving performance [76, 77])—by integrating the Hamiltonian into the optimisation process, scaling beyond few-qubit systems, and avoiding relying on manual reimplementation.

11 Conclusion

In this paper, we presented an interdisciplinary approach that merges SE and ML paradigms to enhance the performance of quantum algorithms using quantum simulators. We designed and implemented a new framework, `AccelerQ`, as a prototype tool to predict near-optimal hyperparameters for quantum algorithms. We evaluated `AccelerQ` on two implementations (ADAPT-QSCI or QCELS), training and deploying relatively small-scale models to improve performance by suggesting better hyperparameters. Our results suggest that the model's predictive ability depends on the Hamiltonian's characteristics rather than solely on a specific implementation.

Beyond optimising complex quantum simulations, `AccelerQ` also provides deeper insights into the underlying physics of the studied systems. For instance, by tuning the number of relevant terms of the Hamiltonians (through coefficient cut-off points), we can identify relevant correlation terms.

Future Work. While our evaluation includes three representative configurations (`FULL-AccelerQ`, `OPT-ML-Only`, `DEFAULT`) for an initial ablation study, we acknowledge the absence of broader empirical comparisons with prior work. Each full configuration requires 3-6 months of computation across multiple CPU machines, which constrained our ability to explore the experimental space exhaustively. Expanding the comparative evaluation to include other methods and variants is a direction for follow-up work.

Besides comparison against similar methods and other variational algorithms' implementations when experimentally possible, this methodology could be extended to inferring more complex chemistry-related terms, such as the ansatz, a key bottleneck in quantum chemistry simulations. Fine-tuning these properties is particularly challenging, as small structural changes can significantly impact the quality of quantum simulations. Addressing this complexity requires optimisation beyond hyperparameters, suggesting that ML could play a broader role in refining quantum algorithm execution at multiple levels. The method presented in this work represents an early step toward this direction and may pave the way for integrating ML and quantum algorithms into more robust and scalable QC applications.

12 Data Availability Statement

Software, setups, and datasets are on Zenodo DOI 10.5281/zenodo.16878135 [10]. The artifact, deemed reusable by the AEC, includes modular code, partial evaluations, full pipelines, pre-trained models, scripts, Docker for reproducibility, and guidance on reproducing results with other QE implementations, in addition to those presented here.

Author Contributions. The authors are listed alphabetically and contributed equally to this work.

Acknowledgments. The authors are listed alphabetically. We thank CloudLab [26] for providing the infrastructure that enabled building the two `QE predictors` for ADAPT-QSCI and QCELS. This work was supported by EPSRC projects VSL-Q (EP/Y005244/1), RoaRQ (Investigation 009), and ModeMCQ (EP/W032635/1), the QAssure project from Innovate UK, and King's Quantum grants from King's College London.

References

[1] Amine Mohamed Aboussalah, Changhun Chi, and Chang-Gun Lee. 2023. Quantum computing reduces systemic risk in financial networks. *Scientific Reports* 13 (March 2023), 3990. https://doi.org/10.1038/s41598-023-30710-z

[2] Giovanni Acampora, Ferdinando Di Martino, Alfredo Massa, Roberto Schiattarella, and Autilia Vitiello. 2023. D-NISQ: A reference model for Distributed Noisy Intermediate-Scale Quantum computers. *Inf. Fusion* 89, C (Jan. 2023), 16–28. https://doi.org/10.1016/j.inffus.2022.08.003

[3] Mohamed Ali al Badri, Edward Linscott, Antoine Georges, Daniel J. Cole, and Cédric Weber. 2020. Superexchange mechanism and quantum many body excitations in the archetypal di-Cu oxo-bridge. *Communications Physics* 3, 1 (2020), 4. https://doi.org/10.1038/s42005-019-0270-1

[4] Amazon Web Services. 2020. Amazon Braket: Developer Guide. https://docs.aws.amazon.com/braket/index.html.

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (Oct. 2019), 505–510. https://www.nature.com/articles/s41586-019-1666-5

[6] Thomas Ayral, Thibaud Louvet, Yiqing Zhou, Cyprien Lambert, E. Miles Stoudenmire, and Xavier Waintal. 2023. Density-Matrix Renormalization Group Algorithm for Simulating Quantum Circuits with a Finite Fidelity. *PRX Quantum* 4 (Apr 2023), 020304. Issue 2. https://doi.org/10.1103/PRXQuantum.4.020304

[7] Charles H. Bennett and Gilles Brassard. 1987. Quantum public key distribution reinvented. *SIGACT News* 18, 4 (jul 1987), 51–53. https://doi.org/10.1145/36068.36070

[8] Charles H. Bennett, Gilles Brassard, Seth Breidbart, and Stephen Wiesner. 1983. Quantum Cryptography, or Unforgeable Subway Tokens. In *Advances in Cryptology*, David Chaum, Ronald L. Rivest, and Alan T. Sherman (Eds.). Springer US, Boston, MA, 267–275.

[9] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. 1993. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.* 70 (Mar 1993), 1895–1899. Issue 13. https://doi.org/10.1103/PhysRevLett.70.1895

[10] Avner Bensoussan, Elena Chachkarova, Karine Even Mendoza, Sophie Fortz, and Connor Lenihan. 2025. *Artifact of AccelerQ: Accelerating Quantum Eigensolvers With Machine Learning on Quantum Simulators.* Zenodo. https://doi.org/10.5281/zenodo.16878135

[11] Harel Berger, Aidan Dakhama, Zishuo Ding, Karine Even-Mendoza, David Kelly, Hector D. Menendez, Rebecca Moussa, and Federica Sarro. 2023. *StableYolo: Optimizing Image Generation for Large Language Models.* Springer, Cham, 133–139.

[12] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. 2022. Noisy intermediate-scale quantum (NISQ) algorithms. *Rev. Mod. Phys.* 94 (Feb 2022), 015004. Issue 1. https://doi.org/10.1103/RevModPhys.94.015004

[13] Gilles Brassard, Peter Høyer, and Alain Tapp. 2016. *Quantum Algorithm for the Collision Problem.* Springer New York, New York, NY, 1662–1664. https://doi.org/10.1007/978-1-4939-2864-4_304

[14] Alexander E. I. Brownlee, James Callan, Karine Even-Mendoza, Alina Geiger, Carol Hanna, Justyna Petke, Federica Sarro, and Dominik Sobania. 2024. Enhancing Genetic Improvement Mutations Using Large Language Models. In *Search-Based Software Engineering*, Paolo Arcaini, Tao Yue, and Erik M. Fredericks (Eds.). Springer Nature Switzerland, Cham, 153–159.

[15] M. Cerezo, Martin Larocca, Diego García-Martín, N. L. Diaz, Paolo Braccia, Enrico Fontana, Manuel S. Rudolph, Pablo Bermejo, Aroosa Ijaz, Supanut Thanasilp, Eric R. Anschuetz, and Zoë Holmes. 2024. Does provable absence of barren plateaus imply classical simulability? Or, why we need to rethink variational quantum computing. arXiv:2312.09121 [quant-ph] https://arxiv.org/abs/2312.09121 preprint.

[16] Gyungmin Cho and Dohun Kim. 2024. Machine learning on quantum experimental data toward solving quantum many-body problems. *Nature Communications* 15, 1 (Aug. 2024), 7552. https://www.nature.com/articles/s41467-024-51932-3 Publisher: Nature Publishing Group.

[17] Elias F Combarro, Alberto Di Meglio Samuel González-Castillo, and Alberto Di Meglio. 2023. *A practical guide to quantum machine learning and quantum optimization.* Packt Publishing, UK.

[18] Connorpl. Accessed: July 4, 2024. QCELS_for_QAGC. https://github.com/Connorpl/QCELS_for_QAGC.
[19] S. Consul-Pacareu, R. Montaño, Kevin Rodriguez-Fernandez, Àlex Corretgé, Esteve Vilella-Moreno, Daniel Casado-Faulí, and Parfait Atchade-Adelomou. 2023. Quantum Machine Learning hyperparameter search. arXiv:2302.10298 [cs.LG] https://arxiv.org/abs/2302.10298 preprint.
[20] Elbio Dagotto. 1994. Correlated electrons in high-temperature superconductors. *Reviews of Modern Physics* 66, 3 (1994), 763–840. https://doi.org/10.1103/RevModPhys.66.763
[21] Aidan Dakhama, Karine Even-Mendoza, William B. Langdon, Héctor D. Menéndez, and Justyna Petke. 2023. SearchGEM5: Towards Reliable Gem5 with Search Based Software Testing and Large Language Models. In *SSBSE 2023, Proceedings (LNCS, Vol. 14415)*. Springer, San Francisco, CA, USA, 160–166. https://doi.org/10.1007/978-3-031-48796-5_14 Best challenge track paper..
[22] Siddharth Dangwal, Gokul Subramanian Ravi, Lennart Maximilian Seifert, Poulami Das, James Sud, and Frederic T. Chong. 2024. COMPASS: Compiler Pass Selection For Improving Fidelity Of NISQ Applications. In *2024 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, IEEE, Los Alamitos, CA, USA, 1–14. https://doi.org/10.1109/ICRC64395.2024.10937002
[23] David Deutsch and Richard Jozsa. 1992. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439, 1907 (1992), 553–558.
[24] Zhiyan Ding and Lin Lin. 2023. Even Shorter Quantum Circuit for Phase Estimation on Early Fault-Tolerant Quantum Computers with Applications to Ground-State Energy Estimation. *PRX Quantum* 4, 2 (May 2023), 020331. Issue 2. https://doi.org/10.1103/PRXQuantum.4.020331 Publisher: American Physical Society.
[25] Zhiyan Ding and Lin Lin. 2023. Simultaneous estimation of multiple eigenvalues with short-depth quantum circuit on early fault-tolerant quantum computers. *Quantum* 7 (Oct. 2023), 1136. https://quantum-journal.org/papers/q-2023-10-11-1136/ Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.
[26] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The design and operation of cloudlab. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 1–14.
[27] Vinicius H. S. Durelli, Rafael S. Durelli, Simone S. Borges, Andre T. Endo, Marcelo M. Eler, Diego R. C. Dias, and Marcelo P. Guimarães. 2019. Machine Learning Applied to Software Testing: A Systematic Mapping Study. *IEEE Transactions on Reliability* 68, 3 (2019), 1189–1212. https://doi.org/10.1109/TR.2019.2892517
[28] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, Los Alamitos, CA, USA, 31–53. https://doi.org/10.1109/ICSE-FOSE59343.2023.00008
[29] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. 2001. A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem. *Science* 292, 5516 (2001), 472–475. https://doi.org/10.1126/science.1057726 arXiv:https://www.science.org/doi/pdf/10.1126/science.1057726
[30] D.V. Fastovets, Yu.I. Bogdanov, B.I. Bantysh, and V.F. Lukichev. 2019. Machine learning methods in quantum computing theory. In *International Conference on Micro- and Nano-Electronics 2018*, Vladimir F. Lukichev and Konstantin V. Rudenko (Eds.). SPIE, Zvenigorod, Russian Federation, 85. https://doi.org/10.1117/12.2522427
[31] Richard P Feynman. 1982. Simulating physics with computers. *International Journal of Theoretical Physics* 21, 6/7 (1982), 467–488.
[32] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics* 29, 5 (2001), 1189–1232.
[33] Ilie-Daniel Gheorghe-Pop, Nikolay Tcholtchev, Tom Ritter, and Manfred Hauswirth. 2020. Quantum DevOps: Towards Reliable and Applicable NISQ Quantum Computing. In *2020 IEEE Globecom Workshops (GC Wkshps*. IEEE, IEEE, Taipei, Taiwan, 1–6. https://doi.org/10.1109/GCWkshps50303.2020.9367411
[34] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
[35] Felix Greiwe, Tom Krüger, and Wolfgang Mauerer. 2023. Effects of Imperfections on Quantum Algorithms: A Software Engineering Perspective. , 31-42 pages. https://doi.org/10.1109/QSW59989.2023.00014
[36] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. 2019. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications* 10, 1 (July 2019), 3007. https://doi.org/10.1038/s41467-019-10988-2
[37] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. ACM, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[38] Thomas Häner, Torsten Hoefler, and Matthias Troyer. 2020. Assertion-based optimization of Quantum programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 133 (Nov. 2020), 20 pages. https://doi.org/10.1145/3428201

[39] Mark Harman. 2012. The role of Artificial Intelligence in Software Engineering. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. IEEE, IEEE, Los Alamitos, CA, USA, 1–6. https://doi.org/10.1109/RAISE.2012.6227961

[40] Mark Harman and Bryan F Jones. 2001. Search-based software engineering. *Information and Software Technology* 43, 14 (2001), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

[41] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. https://doi.org/10.1145/2379776.2379787

[42] Sabrina Herbst, Vincenzo De Maio, and Ivona Brandic. 2024. On Optimizing Hyperparameters for Quantum Neural Networks. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01. IEEE, IEEE, Los Alamitos, CA, USA, 1478–1489. https://doi.org/10.1109/QCE60285.2024.00174

[43] Dylan Herman, Connor Googin, Xin Liu, Yudong Sun, Alexey Galda, Ilya Safro, Marco Pistoia, and Yuri Alexeev. 2023. Quantum computing for finance. *Nature Reviews Physics* 5 (June 2023), 450–465. https://doi.org/10.1038/s42254-023-00603-1

[44] Nils Herrmann, Daanish Arya, Marcus W. Doherty, Angus Mingare, Jason C. Pillay, Florian Preis, and Stefan Prestel. 2023. Quantum utility – definition and assessment of a practical quantum advantage. In *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, IEEE, Los Alamitos, CA, USA, 162–174. https://doi.org/10.1109/QSW59989.2023.00028

[45] Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. 2024. When large language model meets optimization. , 101663 pages. https://doi.org/10.1016/j.swevo.2024.101663

[46] Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. *Physical Review E* 58, 5 (1998), 5355–5363. https://doi.org/10.1103/PhysRevE.58.5355

[47] Chan Gu Kang and Hakjoo Oh. 2023. Modular Component-Based Quantum Circuit Synthesis. *Artifact for paper "Modular Component-Based Quantum Circuit Synthesis"* 7, OOPSLA1 (April 2023), 87:348–87:375. https://doi.org/10.1145/3586039

[48] Keita Kanno, Masaya Kohda, Ryosuke Imai, Sho Koh, Kosuke Mitarai, Wataru Mizukami, and Yuya O. Nakagawa. 2023. Quantum-Selected Configuration Interaction: classical diagonalization of Hamiltonians in subspaces selected by quantum computers. arXiv:2302.11320 [quant-ph] https://arxiv.org/abs/2302.11320 preprint.

[49] A. Yu. Kitaev. 1995. Quantum measurements and the Abelian Stabilizer Problem. arXiv:quant-ph/9511026 [quant-ph] https://arxiv.org/abs/quant-ph/9511026 preprint.

[50] Handy Kurniawan, Laura Rodríguez-Soriano, Daniele Cuomo, Carmen G. Almudever, and Francisco García Herrero. 2024. On the Use of Calibration Data in Error-Aware Compilation Techniques for NISQ Devices. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01. IEEE, Los Alamitos, CA, USA, 338–348. https://doi.org/10.1109/QCE60285.2024.00048

[51] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri, A. Aspuru-Guzik, and A. G. White. 2010. Towards quantum chemistry on a quantum computer. *Nature Chemistry* 2, 2 (2010), 106–111.

[52] Liyi Li, Finn Voichick, Kesha Hietala, Yuxiang Peng, Xiaodi Wu, and Michael Hicks. 2022. Verified compilation of Quantum oracles. *Reproduction Package for "Verified Compilation of Quantum Oracles"* 6, OOPSLA2 (Oct. 2022), 146:589–146:615. https://doi.org/10.1145/3563309

[53] Zikun Li, Jinjun Peng, Yixuan Mei, Sina Lin, Yi Wu, Oded Padon, and Zhihao Jia. 2024. Quarl: A Learning-Based Quantum Circuit Optimizer. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 114 (April 2024), 28 pages. https://doi.org/10.1145/3649831

[54] Hocheol Lim, Doo Hyung Kang, Jeonghoon Kim, Aidan Pellow-Jarman, Shane McFarthing, Rowan Pellow-Jarman, Hyeon-Nae Jeon, Byungdu Oh, June-Koo Kevin Rhee, and Kyoung Tai No. 2024. Fragment molecular orbital-based variational quantum eigensolver for quantum chemistry in the age of quantum computing. *Scientific Reports* 14, 1 (2024), 2422.

[55] William G. Macready and David H. Wolpert. 1995. *No Free Lunch Theorems for Search.* Technical Report 1995-02-010. Santa Fe Institute. https://sfi-edu.s3.amazonaws.com/sfi-edu/production/uploads/sfi-com/dev/uploads/filer/3c/34/3c34c50b-4ea5-4715-b9eb-813fb7085504/95-02-010.pdf SFI WORKING PAPER.

[56] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Rev. Mod. Phys.* 92 (Mar 2020), 015003. Issue 1. https://doi.org/10.1103/RevModPhys.92.015003

[57] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. 2018. Barren plateaus in quantum neural network training landscapes. *Nature Communications* 9, 1 (16 Nov 2018), 4812. https://doi.org/10.1038/s41467-018-07090-4

[58] Sara Ayman Metwalli and Rodney Van Meter. 2022. A Tool For Debugging Quantum Circuits. http://arxiv.org/abs/2205.01899 preprint.
[59] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Thomas Häner. 2020. Enabling accuracy-aware Quantum compilers using symbolic resource estimation. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 130 (Nov. 2020), 26 pages. https://doi.org/10.1145/3428198
[60] Andriy Miranskyy and Lei Zhang. 2019. On testing quantum programs. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results* (Montreal, Quebec, Canada) *(ICSE-NIER '19)*. IEEE, IEEE, Los Alamitos, CA, USA, 57–60. https://doi.org/10.1109/ICSE-NIER.2019.00023
[61] Charles Moussa, Yash J. Patel, Vedran Dunjko, Thomas Bäck, and Jan N. van Rijn. 2024. Hyperparameter importance and optimization of quantum neural networks across small datasets. *Machine Learning* 113, 4 (01 Apr 2024), 1941–1966. https://doi.org/10.1007/s10994-023-06389-8
[62] Patrick Mulligan, Clemens Masteran, Adam Finkelstein, Peter D. Johnson, and Oleksandr Kyriienko. 2022. Numerical simulations of noisy quantum circuits for computational chemistry. *Journal of Materials Science: Materials Theory* 6, 1 (2022), 6. https://doi.org/10.1186/s41313-022-00047-7
[63] Yuya O. Nakagawa, Masahiko Kamoshita, Wataru Mizukami, Shotaro Sudo, and Yu ya Ohnishi. 2023. ADAPT-QSCI: Adaptive Construction of Input State for Quantum-Selected Configuration Interaction. arXiv:2311.01105 [quant-ph] https://arxiv.org/abs/2311.01105 preprint.
[64] Kim A. Nicoli, Luca Johannes Wagner, and Lena Funcke. 2025. Machine-Learning-Enhanced Optimization of Noise-Resilient Variational Quantum Eigensolvers. *PoS* LATTICE2024 (2025), 417. arXiv:2501.17689 [quant-ph] https://pos.sissa.it/466/417/pdf preprint.
[65] Michael A Nielsen et al. 2005. The Fermionic canonical commutation relations and the Jordan-Wigner transform. *School of Physical Sciences The University of Queensland* 59 (2005), 75.
[66] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, UK.
[67] Kyungjoo Noh, Liang Jiang, and Bill Fefferman. 2020. Efficient classical simulation of noisy random quantum circuits in one dimension. *Quantum* 4 (Sept. 2020), 318. https://doi.org/10.22331/q-2020-09-11-318
[68] Matteo Paltenghi and Michael Pradel. 2022. Bugs in Quantum computing platforms: an empirical study. *Reproduction Package for "Bugs in Quantum Computing Platforms: An Empirical Study"* 6, OOPSLA1 (April 2022), 86:1–86:27. https://doi.org/10.1145/3527330
[69] Anouk Paradis, Jasper Dekoninck, Benjamin Bichsel, and Martin Vechev. 2024. Synthetiq: Fast and Versatile Quantum Circuit Synthesis. *Reproduction Package for the Article "Synthetiq: Fast and Versatile Quantum Circuit Synthesis"* 8, OOPSLA1 (April 2024), 96:55–96:82. https://doi.org/10.1145/3649813
[70] David Peral-García, Juan Cruz-Benito, and Francisco José García-Peñalvo. 2024. Systematic literature review: Quantum machine learning and its applications. *Computer Science Review* 51 (Feb. 2024), 100619. https://doi.org/10.1016/j.cosrev.2024.100619
[71] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a quantum processor. *Nature Communications* 5, 1 (July 2014), 4213. http://arxiv.org/abs/1304.3061 preprint.
[72] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79
[73] Qiskit Development Team. 2023. Qiskit: An Open-source Framework for Quantum Computing. https://qiskit.org/. Accessed: 2024-07-01.
[74] Google AI Quantum and Collaborators. 2021. Cirq: A Python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits. https://github.com/quantumlib/Cirq.
[75] QunaSys. Accessed: July 4, 2024. Homepage. https://qunasys.com/en/.
[76] QunaSys. February 1, 2023. Quantum Algorithm Grand Challenge 2023 (QAGC2023). https://github.com/QunaSys/quantum-algorithm-grand-challenge-2023.
[77] QunaSys. February 1, 2024. Quantum Algorithm Grand Challenge 2024 (QAGC2024). https://github.com/QunaSys/quantum-algorithm-grand-challenge-2024.
[78] Somayeh Bakhtiari Ramezani, Alexander Sommers, Harish Kumar Manchukonda, Shahram Rahimi, and Amin Amirlatifi. 2020. Machine Learning Algorithms in Quantum Computing: A Survey. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, IEEE, Los Alamitos, CA, USA, 1–8. https://ieeexplore.ieee.org/document/9207714 ISSN: 2161-4407.
[79] Ariel Rosenfeld, Odaya Kardashov, and Orel Zang. 2018. Automation of Android applications functional testing using machine learning activities classification. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems* (Gothenburg, Sweden) *(MOBILESoft '18)*. ACM, New York, NY, USA, 122–132. https://doi.org/10.1145/3197231.3197241

[80] Naoto Sato and Ryota Katsube. 2023. Locating Buggy Segments in Quantum Program Debugging. http://arxiv.org/abs/2309.04266 preprint.
[81] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172
[82] Ivan Šupić and Joseph Bowles. 2020. Self-testing of quantum systems: a review. *Quantum* 4 (Sept. 2020), 337. https://doi.org/10.22331/q-2020-09-30-337 arXiv:1904.10042 [quant-ph].
[83] Masuo Suzuki. 1976. Generalized Trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems. *Communications in Mathematical Physics* 51, 2 (01 Jun 1976), 183–190. https://doi.org/10.1007/BF01609348
[84] Attila Szabo and Neil S. Ostlund. 1996. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory.* Dover Publications, Mineola, NY, USA.
[85] Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: push-button verification for the qiskit Quantum compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 641–656. https://doi.org/10.1145/3519939.3523431
[86] Yanxian Tao, Xiongzhi Zeng, Yi Fan, Jie Liu, Zhenyu Li, and Jinlong Yang. 2022. Exploring Accurate Potential Energy Surfaces via Integrating Variational Quantum Eigensolver with Machine Learning. *The Journal of Physical Chemistry Letters* 13, 28 (2022), 6420–6426. https://doi.org/10.1021/acs.jpclett.2c01738
[87] Jules Tilly, Hongxiang Chen, Shuxiang Cao, Dario Picozzi, Kanav Setia, Ying Li, Edward Grant, Leonard Wossnig, Ivan Rungger, George H. Booth, and Jonathan Tennyson. 2022. The Variational Quantum Eigensolver: a review of methods and best practices. *Physics Reports* 986 (2022), 1–128. https://doi.org/10.1016/j.physrep.2022.08.003 The Variational Quantum Eigensolver: a review of methods and best practices.
[88] H. F. Trotter. 1959. On the Product of Semi-Groups of Operators. *Proc. Amer. Math. Soc.* 10, 4 (1959), 545–551. http://www.jstor.org/stable/2033649
[89] Muhammad Junaid Umer and Muhammad Imran Sharif. 2022. A Comprehensive Survey on Quantum Machine Learning and Possible Applications. *Int. J. E-Health Med. Commun.* 13, 5 (Dec. 2022), 1–17. https://doi.org/10.4018/IJEHMC.315730
[90] Hristo Venev, Timon Gehr, Dimitar Dimitrov, and Martin Vechev. 2024. Modular Synthesis of Efficient Quantum Uncomputation. *Proceedings of the ACM on Programming Languages* 8, OOPSLA2 (Oct. 2024), 2097–2124. https://doi.org/10.1145/3689785
[91] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936. https://doi.org/10.1109/TSE.2024.3368208
[92] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. 2023. Machine/Deep Learning for Software Engineering: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 49, 3 (2023), 1188–1231. https://doi.org/10.1109/TSE.2022.3173346
[93] Yunfei Wang and Junyu Liu. 2024. A comprehensive review of quantum machine learning: from NISQ to fault tolerance. *Reports on Progress in Physics* 87, 11 (oct 2024), 116402. https://doi.org/10.1088/1361-6633/ad7f69
[94] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A systematic literature review on the use of deep learning in software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–58.
[95] Christopher A. Welty and Peter G. Selfridge. 1997. Artificial Intelligence and Software Engineering: Breaking the Toy Mold. *Automated Software Engineering* 4, 3 (1997), 255–270. https://doi.org/10.1023/A:1008662625094
[96] Xanadu. 2023. PennyLane: Quantum machine learning, automatic differentiation, and optimization of hybrid quantum-classical computations. https://pennylane.ai/. Accessed: 2024-07-01.
[97] xgboost developers. 2022. XGBoost Tutorials. https://xgboost.readthedocs.io/en/stable/tutorials/model.html.
[98] Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proc. ACM Program. Lang.* 7, PLDI, Article 140 (June 2023), 25 pages. https://doi.org/10.1145/3591254
[99] Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: superoptimization of Quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 625–640. https://doi.org/10.1145/3519939.3523433
[100] Peng Yan, Hanru Jiang, and Nengkun Yu. 2022. On incorrectness logic for Quantum programs. *Proc. ACM Program. Lang.* 6, OOPSLA1 (April 2022), 72:1–72:28. https://doi.org/10.1145/3527316
[101] Yanming Yang, Xin Xia, David Lo, and John Grundy. 2022. A Survey on Deep Learning for Software Engineering. *ACM Comput. Surv.* 54, 10s, Article 206 (Sept. 2022), 73 pages. https://doi.org/10.1145/3505243

[102] Qianjun Yao, Qun Ji, Xiaopeng Li, Yehui Zhang, Xinyu Chen, Ming-Gang Ju, Jie Liu, and Jinlan Wang. 2024. Machine Learning Accelerates Precise Excited-State Potential Energy Surface Calculations on a Quantum Computer. *The Journal of Physical Chemistry Letters* 15, 27 (2024), 7061–7068. https://doi.org/10.1021/acs.jpclett.4c01445

[103] Tong Yu and Hong Zhu. 2020. Hyper-Parameter Optimization: A Review of Algorithms and Applications. arXiv:2003.05689 [cs.LG] https://arxiv.org/abs/2003.05689 preprint.

[104] Du Zhang and Jeffrey J. P. Tsai. 2003. Machine Learning and Software Engineering. *Software Quality Journal* 11, 2 (2003), 87–119. https://doi.org/10.1023/A:1023760326768

[105] Meng Zhang, Chao Wang, Shaojun Dong, Hao Zhang, Yongjian Han, and Lixin He. 2022. Entanglement entropy scaling of noisy random quantum circuits in two dimensions. *Phys. Rev. A* 106 (Nov 2022), 052430. Issue 5. https://doi.org/10.1103/PhysRevA.106.052430

[106] Yiqing Zhou, E. Miles Stoudenmire, and Xavier Waintal. 2020. What Limits the Simulation of Quantum Computers? *Phys. Rev. X* 10 (Nov 2020), 041038. Issue 4. https://doi.org/10.1103/PhysRevX.10.041038

Received 2025-03-26; accepted 2025-08-12