A Numerical Optimization Study of the Variational Quantum Eigensolver

Sara Gholamhoseinian[1]
[1]University of Massachusetts Dartmouth, North Dartmouth 02747, MA, USA

Abstract

The Variational Quantum Eigensolver (VQE) is a promising hybrid quantum-classical algorithm for computing ground state energies on near-term quantum devices. While the quantum circuit architecture has been extensively studied, the choice of classical optimization strategy remains a critical factor affecting convergence speed, accuracy, and robustness. This work presents a systematic comparison of optimization methods for VQE, evaluating gradient estimation techniques (finite difference and parameter shift rule), optimization algorithms (gradient descent, momentum, Adam, and quantum natural gradient), and step-size strategies (constant and decaying schedules). Using a 3-qubit transverse-field Ising model as a benchmark, we assess optimizer performance through convergence analysis, final energy error, and robustness over random initializations. Our results show that quantum natural gradient achieves the highest accuracy with final errors below 10^{-5}, while Adam demonstrates superior robustness with a 50% success rate compared to 23% for standard gradient descent. However, when accounting for computational cost, the parameter shift rule requires twice as many energy evaluations per iteration as finite difference, creating a trade-off between gradient accuracy and efficiency. We find that larger constant step sizes accelerate convergence but reduce stability, while decaying schedules balance these concerns. These findings provide practical guidance for selecting optimization strategies in VQE implementations, particularly for applications where computational budget and solution quality must be carefully balanced.

1 Introduction

Computing the ground state energy of quantum systems is a fundamental problem in quantum chemistry, materials science, and condensed matter physics. The minimum eigenvalue of a system's Hamiltonian determines crucial properties such as molecular binding energies, reaction pathways, and material stability [1]. Classical computational methods for this task, such as exact diagonalization or coupled-cluster approaches, face exponential scaling with system size, limiting their applicability to small molecules or requiring severe approximations for larger systems.

Quantum computers promise exponential speedup for certain quantum simulation tasks. The quantum phase estimation (QPE) algorithm can theoretically determine eigenvalues with high precision [2]. However, QPE requires circuit depths that scale unfavorably with desired accuracy, demanding thousands of coherent quantum gates. Such deep circuits exceed the capabilities of current noisy intermediate-scale quantum (NISQ) devices, where decoherence and gate errors accumulate rapidly, rendering QPE impractical for near-term applications.

The Variational Quantum Eigensolver (VQE), introduced by Peruzzo et al. [1], offers an alternative approach tailored to NISQ constraints. VQE employs a hybrid quantum-classical architecture: a parameterized quantum circuit prepares trial states, quantum hardware measures expectation values, and a classical optimizer iteratively refines parameters to minimize the energy. By leveraging shallow quantum circuits and offloading optimization to classical computers, VQE circumvents the coherence time limitations that plague QPE. Since its introduction, VQE has been successfully

applied to small molecules [3], demonstrated on various quantum hardware platforms, and extended to excited states and other quantum chemistry problems.

While the quantum aspects of VQE such as ansatz design, circuit depth reduction, and measurement strategies, have received considerable attention, the classical optimization component remains comparably understudied. Yet the optimizer's performance directly determines VQE's practical efficiency: convergence speed affects total runtime, accuracy determines solution quality, and robustness influences reliability across different problem instances. The choice of gradient estimation method introduces additional trade-offs, as quantum circuits enable exact gradient computation via the parameter shift rule at the cost of additional measurements, while approximate finite difference methods require fewer evaluations but introduce errors.

This work provides a systematic empirical study of optimization strategies for VQE, addressing three key questions:

1. How do different gradient estimation techniques (finite difference vs. parameter shift rule) compare when accounting for their computational cost?
2. Which optimization algorithms (gradient descent, momentum, Adam, quantum natural gradient) achieve the best balance of convergence speed, accuracy, and robustness?
3. What impact do step-size strategies (constant vs. decaying schedules) have on optimizer performance?

We evaluate these methods on a 3-qubit transverse-field Ising model using a hardware-efficient ansatz with 9 variational parameters. Our benchmark assesses convergence trajectories, final energy errors relative to exact diagonalization, and statistical robustness over 30 random initializations. By analyzing performance in terms of both iteration count and total energy evaluations, we provide practical guidance for practitioners implementing VQE on real quantum hardware where measurement budget is constrained.

The remainder of this paper is organized as follows. Section 2 reviews the variational principle and VQE formalism. Section 3 describes our experimental setup, gradient methods, optimizers, and evaluation metrics. Section 4 presents comparative performance analysis. Section 5 interprets these findings and discusses practical implications. Section 6 concludes with recommendations for VQE optimization and future research directions.

2 Theoretical Background

2.1 The Variational Principle

The variational method provides a rigorous upper bound on the ground state energy of quantum systems. Consider a Hermitian operator H representing a system Hamiltonian. By the spectral theorem, H possesses real eigenvalues $\{\lambda_i\}$ with corresponding eigenstates $\{|\psi_i\rangle\}$ forming a complete orthonormal basis. The operator admits the spectral decomposition

$$H = \sum_{i=1}^{N} \lambda_i |\psi_i\rangle\langle\psi_i|, \tag{1}$$

where $\lambda_{\min} = \min_i \lambda_i$ denotes the ground state energy.

For an arbitrary normalized quantum state $|\psi\rangle$, the expectation value of H can be expressed as

$$\langle H\rangle_\psi = \langle\psi|H|\psi\rangle = \sum_{i=1}^{N} \lambda_i |\langle\psi_i|\psi\rangle|^2. \tag{2}$$

Since $|\langle\psi_i|\psi\rangle|^2 \geq 0$ represents the probability of measuring state $|\psi_i\rangle$ and $\sum_i |\langle\psi_i|\psi\rangle|^2 = 1$ by normalization, this expectation value is a convex combination of eigenvalues. Consequently,

$$\lambda_{\min} \leq \langle\psi|H|\psi\rangle \quad \text{for all } |\psi\rangle, \tag{3}$$

with equality achieved if and only if $|\psi\rangle = |\psi_{\min}\rangle$, the ground state. This fundamental inequality, known as the **variational principle** [4], guarantees that any trial wavefunction provides an upper bound on the ground state energy. By systematically varying the trial state and minimizing the expectation value, one can obtain arbitrarily accurate approximations to $\lambda_{\min}$.

2.2 Variational Quantum Eigensolver

VQE implements the variational principle through a hybrid quantum-classical architecture [1]. A parameterized quantum circuit, characterized by a unitary operator $U(\theta)$ with parameters $\theta \in \mathbb{R}^d$, acts on an initial state $|\psi_0\rangle$ (typically $|0\rangle^{\otimes n}$) to produce the trial state

$$|\psi(\theta)\rangle = U(\theta)|\psi_0\rangle. \tag{4}$$

The variational energy becomes a function of the parameters:

$$E(\theta) = \langle\psi(\theta)|H|\psi(\theta)\rangle \geq \lambda_{\min}. \tag{5}$$

The VQE workflow consists of three iterative steps: (1) the quantum processor prepares $|\psi(\theta)\rangle$ and measures expectation values of Hamiltonian terms, (2) a classical computer computes $E(\theta)$ from measurement outcomes, and (3) a classical optimizer updates θ to minimize $E(\theta)$. This loop continues until convergence, ideally yielding θ^* such that $E(\theta^*) \approx \lambda_{\min}$.

Ansatz Design. The choice of parameterized circuit $U(\theta)$ critically affects VQE performance. The circuit must be expressive enough to approximate the ground state while remaining implementable on NISQ hardware with limited coherence times. Hardware-efficient ansatzes [5] prioritize shallow circuit depth using native gate sets, trading some expressivity for practical realizability. In this work, we employ the RealAmplitudes ansatz, a hardware-efficient variational form consisting of layers of single-qubit R_y rotations followed by entangling gates, providing a balance between expressiveness and circuit depth.

Gradient-Based Optimization. Minimizing $E(\theta)$ requires computing or approximating gradients $\nabla_\theta E(\theta)$. Two primary approaches exist:

Finite Difference (FD) approximates the partial derivative via

$$\frac{\partial E}{\partial \theta_i} \approx \frac{E(\theta + \epsilon e_i) - E(\theta)}{\epsilon}, \tag{6}$$

where e_i is the i-th unit vector and ϵ is a small perturbation. This requires $d+1$ energy evaluations per gradient but introduces numerical error scaling as $O(\epsilon)$.

Parameter Shift Rule (PS) [6, 7] exploits the structure of quantum circuits to compute exact gradients. For parameterized gates with specific generator properties, the exact derivative is

$$\frac{\partial E}{\partial \theta_i} = \frac{E(\theta + \frac{\pi}{2} e_i) - E(\theta - \frac{\pi}{2} e_i)}{2}. \tag{7}$$

While exact, this method requires $2d$ energy evaluations per gradient, doubling the measurement cost compared to finite difference.

The choice between FD and PS involves trading gradient accuracy against computational cost, a trade-off that becomes critical when quantum circuit evaluations dominate runtime. Moreover, the

gradient information must be utilized effectively by the optimization algorithm. Standard gradient descent updates parameters as $\theta_{k+1} = \theta_k - \eta_k \nabla E(\theta_k)$, where η_k is the step size. However, more sophisticated optimizers can accelerate convergence by incorporating momentum, adaptive learning rates, or geometric information about the parameter space. The efficiency and reliability of these classical optimization routines play a central role in VQE's overall performance, motivating our systematic comparison of gradient methods, optimization algorithms, and step-size strategies.

3 Methods

3.1 Problem Setup: Transverse-Field Ising Model

We benchmark optimization strategies using a 3-qubit transverse-field Ising model, a paradigmatic system in quantum many-body physics exhibiting nontrivial ground state structure. The Hamiltonian is given by

$$H = -\sum_{i=1}^{3} h_i Z_i - \sum_{i<j} J_{ij} Z_i Z_j, \tag{8}$$

where Z_i denotes the Pauli-Z operator on qubit i, h_i represents local magnetic fields, and J_{ij} encodes spin-spin couplings. For our experiments, we consider a one-dimensional chain topology with nearest-neighbor interactions:

$$J = \begin{pmatrix} 0 & 1.0 & 0 \\ 1.0 & 0 & 1.0 \\ 0 & 1.0 & 0 \end{pmatrix}, \quad h = \begin{pmatrix} 0.5 \\ -0.5 \\ 0.2 \end{pmatrix}. \tag{9}$$

The exact ground state energy, obtained via classical diagonalization of the 8×8 Hamiltonian matrix, is $E_{\text{exact}} = -2.2$, serving as our benchmark target.

Variational Ansatz. We employ the hardware-efficient RealAmplitudes ansatz [5] with 2 repetition layers and full entanglement. This ansatz alternates layers of parameterized single-qubit $R_y(\theta)$ rotations with entangling CX gates connecting all qubit pairs. For $n = 3$ qubits and $r = 2$ repetitions, the total number of variational parameters is $d = n(r + 1) = 9$. The ansatz circuit structure is:

$$U(\theta) = \left[\prod_{i<j} \text{CX}_{ij}\right] \left[\prod_{k=1}^{3} R_y(\theta_k)\right] \times (\text{repeat}), \tag{10}$$

applied to the initial state $|000\rangle$. This choice balances expressiveness and circuit depth, making it suitable for NISQ implementations while providing sufficient flexibility to approximate the ground state.

3.2 Gradient Estimation Methods

We compare two gradient estimation techniques that differ in accuracy and computational cost:

Finite Difference (FD). The gradient is approximated numerically using forward differences:

$$[\nabla E(\theta)]_i \approx \frac{E(\theta + \epsilon e_i) - E(\theta)}{\epsilon}, \tag{11}$$

with perturbation $\epsilon = 10^{-4}$. This requires $d + 1 = 10$ energy evaluations per gradient: one for the reference point $E(\theta)$ and one for each perturbed parameter. The method is simple and general but introduces approximation error of order $O(\epsilon)$.

Parameter Shift Rule (PS). Exploiting the analytic properties of parameterized quantum gates, exact gradients are computed as [6]:

$$[\nabla E(\theta)]_i = \frac{E(\theta + \frac{\pi}{2}e_i) - E(\theta - \frac{\pi}{2}e_i)}{2}. \tag{12}$$

This requires $2d = 18$ energy evaluations per gradient (two evaluations per parameter). While exact, PS incurs an 80% cost increase relative to FD when measured by total circuit evaluations.

The trade-off between gradient accuracy and measurement budget is central to our analysis: we evaluate optimizer performance both by iteration count (favoring FD) and total energy evaluations (normalizing for computational cost).

3.3 Optimization Algorithms

We systematically compare four optimization algorithms representing different design philosophies:

Gradient Descent (GD). The baseline first-order method updates parameters as

$$\theta_{k+1} = \theta_k - \eta_k \nabla E(\theta_k), \tag{13}$$

where η_k is the step size at iteration k. GD provides a reference for evaluating more sophisticated methods.

Momentum Gradient Descent (Momentum). Incorporates velocity accumulation to accelerate convergence [8]:

$$v_{k+1} = \beta v_k + \nabla E(\theta_k), \quad \theta_{k+1} = \theta_k - \eta_k v_{k+1}, \tag{14}$$

with momentum coefficient $\beta = 0.9$. The velocity term v_k accumulates gradient history, enabling faster traversal of flat regions and reducing oscillations in steep directions.

Adam. An adaptive learning rate optimizer combining momentum and per-parameter scaling [9]:

$$m_{k+1} = \beta_1 m_k + (1 - \beta_1)\nabla E(\theta_k), \quad v_{k+1} = \beta_2 v_k + (1 - \beta_2)[\nabla E(\theta_k)]^2, \tag{15}$$

$$\theta_{k+1} = \theta_k - \eta \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}} + \epsilon}, \tag{16}$$

where $\hat{m} = m/(1 - \beta_1^k)$ and $\hat{v} = v/(1 - \beta_2^k)$ are bias-corrected estimates. We use standard hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and learning rate $\eta = 0.02$. Adam adapts step sizes individually for each parameter, providing robustness to poorly scaled problems.

Quantum Natural Gradient (QNG). Incorporates parameter space geometry via the quantum Fisher information matrix [10]:

$$\theta_{k+1} = \theta_k - \eta_k (F(\theta_k) + \lambda I)^{-1} \nabla E(\theta_k), \tag{17}$$

where $F_{ij} = \mathrm{Re}[\langle \partial_i \psi | \partial_j \psi \rangle - \langle \partial_i \psi | \psi \rangle \langle \psi | \partial_j \psi \rangle]$ and regularization $\lambda = 10^{-2}$ ensures numerical stability. The QFIM is approximated numerically using finite differences of the statevector with $\epsilon = 10^{-6}$. QNG accounts for the metric structure of the quantum state manifold, potentially accelerating convergence in poorly conditioned landscapes.

3.4 Step-Size Strategies

We investigate two step-size scheduling approaches:

Constant Step Size. A fixed learning rate $\eta_k = \eta$ for all iterations. We test $\eta \in \{0.02, 0.05, 0.10\}$ to examine sensitivity to this hyperparameter.

Decaying Step Size. An adaptive schedule that reduces the step size over time:

$$\eta_k = \frac{\eta_0}{1 + \text{decay} \cdot k}, \tag{18}$$

with initial rate $\eta_0 = 0.10$ and decay factor 0.05. This strategy allows aggressive initial progress while ensuring asymptotic convergence.

Note that Adam internally adapts its effective learning rate, making it less sensitive to the choice of η compared to gradient-based methods.

3.5 Evaluation Metrics and Experimental Protocol

Performance Metrics. We assess optimizer performance using three criteria:

1. *Final energy error*: $\Delta E = E(\theta_{\text{final}}) - E_{\text{exact}}$, measuring solution accuracy.
2. *Convergence rate*: Iterations required to achieve $\Delta E < 10^{-3}$.
3. *Computational cost*: Total energy evaluations, accounting for gradient method overhead (10 evaluations/iteration for FD, 18 for PS).

Experimental Design. For each optimizer-gradient combination, we perform:

- **Single-run analysis**: Initialize parameters uniformly at random $\theta_0 \sim \mathcal{U}(0, 2\pi)^9$ with seed 42, run for maximum 200 iterations, and record full energy trajectory.
- **Robustness analysis**: Repeat optimization from 30 independent random initializations (seeds 0–29) to assess statistical reliability. We define *success rate* as the fraction of runs achieving $\Delta E < 10^{-3}$.

All optimizers use a convergence tolerance $|E_{k+1} - E_k| < 10^{-6}$ for early stopping. Energy evaluations are performed via statevector simulation (noiseless), isolating optimizer performance from hardware noise effects. This controlled setting enables fair comparison of optimization strategies before deployment on physical quantum devices. Table 1 summarizes all hyperparameters and experimental settings used in this study.

4 Results

We present a systematic evaluation of optimization strategies for VQE on the 3-qubit transverse-field Ising model with exact ground state energy $E_{\text{exact}} = -2.2$. Results are organized by research question and visualized through convergence trajectories, error analysis, and statistical robustness metrics.

4.1 Gradient Estimation Methods

Figure 1 compares finite difference (FD) and parameter shift (PS) gradients using standard gradient descent with constant step size $\eta = 0.05$. The left panel shows that both methods converge to similar final energies around -1.8, with parameter shift achieving marginally lower error. The energy trajectories are nearly indistinguishable, demonstrating that FD's approximation error ($\epsilon = 10^{-4}$) is sufficiently accurate for this 9-parameter optimization problem.

The right panel quantifies the absolute energy difference $|E_{\text{FD}} - E_{\text{PS}}|$ over iterations. The difference oscillates between 10^{-9} and 10^{-5}, remaining several orders of magnitude smaller than the error-to-exact. This confirms that both gradient estimation techniques produce nearly equivalent optimization trajectories for practical purposes.

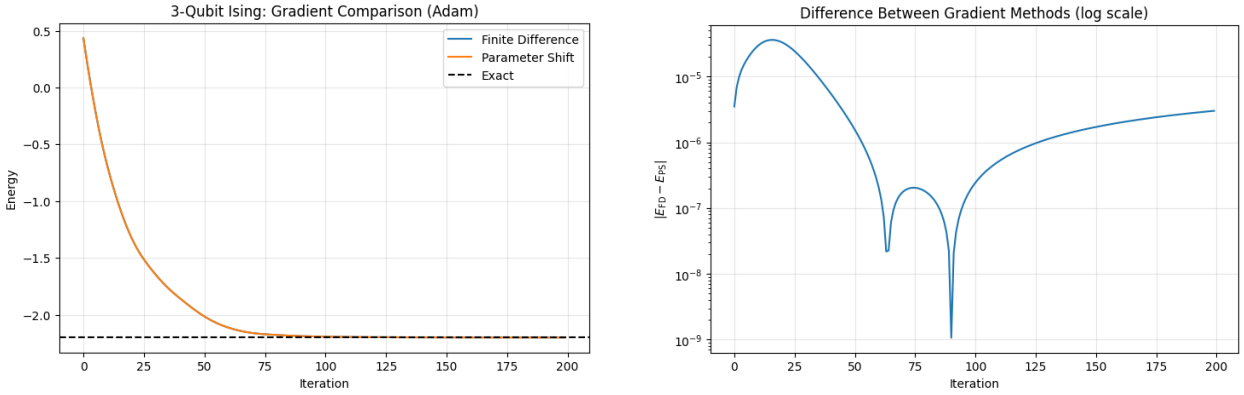

Figure 1: Gradient method comparison. Left: Energy convergence for FD and PS showing nearly overlapping trajectories. Right: Absolute difference between methods remains below 10^{-5}, confirming gradient approximation error is negligible.

While gradient quality is comparable, computational cost differs significantly. Figure 2 plots error versus total energy evaluations, normalizing for the fact that PS requires 18 evaluations per iteration compared to FD's 10 evaluations, an 80% overhead. When measured by total circuit executions rather than iterations, FD achieves comparable accuracy at substantially lower cost. Both methods reach similar error floors ($\sim 4 \times 10^{-4}$), but FD requires approximately 1000 evaluations compared to PS's 2000 evaluations for equivalent convergence.

Key Finding: While parameter shift provides exact gradients, its doubled evaluation cost eliminates the accuracy advantage when measured by computational budget. Finite difference offers competitive performance at 44% lower cost, making it preferable for measurement-limited NISQ implementations.

4.2 Optimizer Performance

We compare four optimization algorithms, including gradient descent (GD), momentum, Adam, and quantum natural gradient (QNG), across both gradient estimation methods. All experiments use constant step size $\eta = 0.05$ for GD, Momentum, and QNG, while Adam uses its default learning rate $\eta = 0.02$.

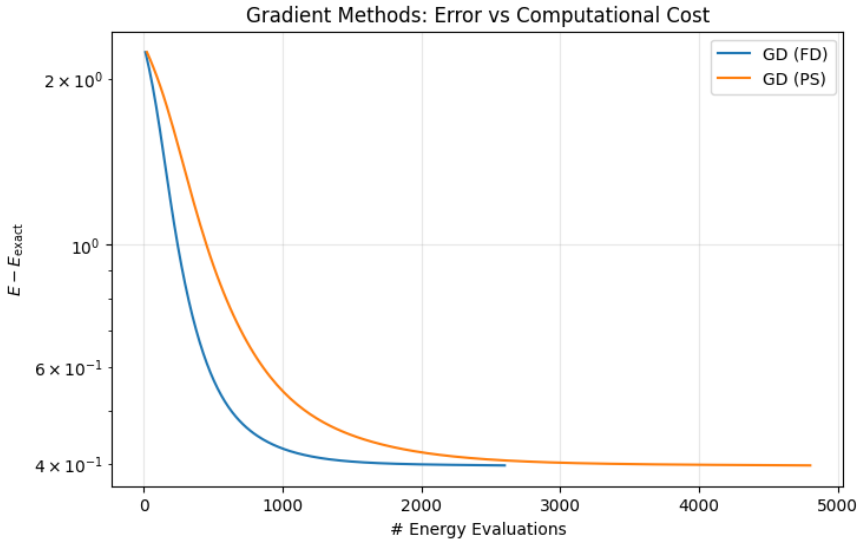

Figure 2: Error vs. total energy evaluations, accounting for computational cost. FD reaches similar accuracy as PS while requiring only 56% of the circuit executions (10 vs. 18 evaluations per gradient).

4.2.1 Finite Difference Gradients

Figure 3 presents optimizer performance with finite difference gradients through two complementary views. The left panel shows raw energy convergence trajectories, while the right panel plots error relative to the exact ground state on a logarithmic scale.

From the energy trajectories (left), QNG (red) achieves the most rapid convergence, reaching the exact ground state energy $E = -2.2$ within approximately 25 iterations. The QNG trajectory is remarkably smooth and monotonic, demonstrating the effectiveness of geometric preconditioning via the Fisher information matrix.

Momentum (orange) and Adam (green) show similar early convergence, both descending rapidly in the first 20 iterations. However, their later behavior differs: Momentum exhibits oscillations around energy -1.8 before slowly approaching the ground state, while Adam maintains steadier descent. Standard GD (blue) demonstrates the slowest convergence, plateauing around -1.8 with minimal progress after 75 iterations.

The error-to-exact plot (right) quantifies these differences more precisely. QNG achieves errors below 10^{-5} by iteration 150, representing nearly exact convergence. Adam reaches approximately 10^{-2} error, while Momentum and GD plateau at higher errors around 10^{-1} to 1. This logarithmic view clearly illustrates QNG's superior asymptotic performance.

4.2.2 Parameter Shift Gradients

With parameter shift gradients (Figure 4), the relative performance ranking remains consistent. The energy trajectories (left) show QNG again demonstrating superior convergence, reaching the exact ground state most rapidly with a smooth trajectory throughout optimization.

The error-to-exact plot (right) reveals that the more accurate PS gradients provide modest improvements across all optimizers. Adam's convergence shows slight improvement compared to FD, reaching approximately 10^{-2} error by iteration 200. Momentum and GD show similar behavior to the FD case, maintaining their characteristic convergence profiles.

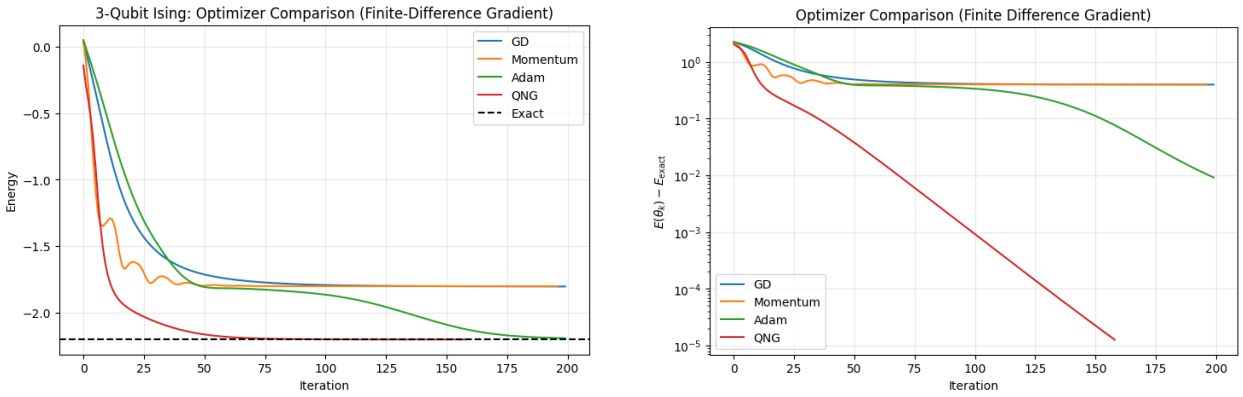

Figure 3: Optimizer comparison using finite difference gradients. Left: Energy convergence showing QNG's rapid descent to exact ground state. Right: Error-to-exact on log scale quantifying final accuracy differences. QNG achieves $< 10^{-5}$ error, while Adam reaches $\sim 10^{-2}$.

Crucially, the performance hierarchy is unchanged: QNG achieves errors below 10^{-5}, Adam reaches $\sim 10^{-2}$, while Momentum and standard GD lag at higher errors. This consistency across gradient methods demonstrates that optimizer architecture dominates performance; the algorithmic differences between QNG's geometric information, Adam's adaptive scaling, and GD's basic descent matter far more than gradient estimation precision for this problem scale.

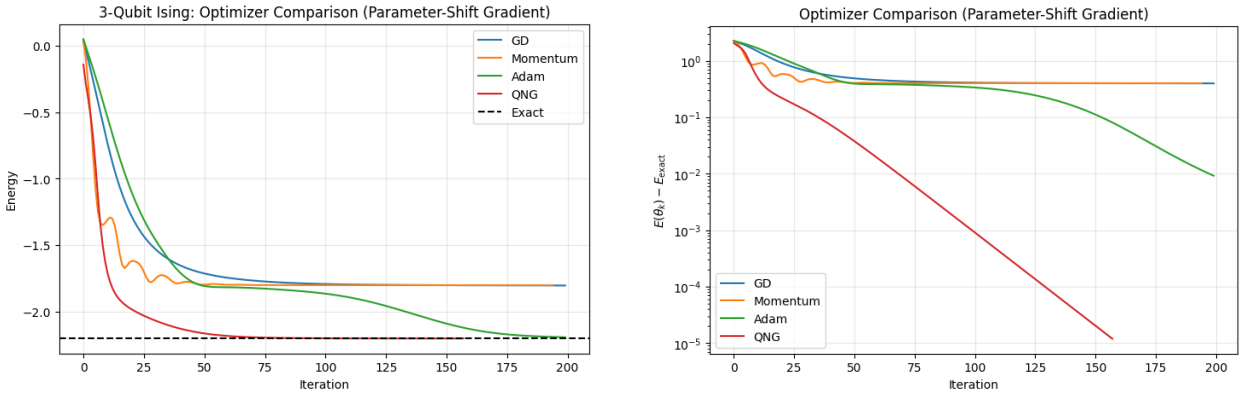

Figure 4: Optimizer comparison using parameter shift gradients. Left: Energy convergence trajectories. Right: Error-to-exact showing QNG achieves $< 10^{-5}$ while Adam reaches $\sim 10^{-2}$ with exact gradients. Performance hierarchy remains consistent with FD results.

Key Finding: QNG consistently achieves the fastest and most accurate convergence (errors $< 10^{-5}$) across gradient methods, validating geometric optimization for VQE. Adam provides a practical alternative with stable convergence and no Fisher matrix overhead. The choice of optimizer has substantially greater impact on performance than the choice of gradient estimation method: all optimizers show similar relative behavior with both FD and PS, confirming that algorithmic design dominates over gradient precision.

4.3 Robustness Analysis

Statistical reliability across random initializations is crucial for practical VQE deployment. Figure 5 shows the distribution of final errors over 30 independent optimization runs using parameter shift gradients, with each run starting from a different random initialization $\theta_0 \sim \mathcal{U}(0, 2\pi)^9$.

QNG demonstrates exceptional robustness with the tightest error distribution. The median error is approximately 2×10^{-5}, with the interquartile range spanning roughly 10^{-5} to 3×10^{-5}. Only one outlier appears above 10^{-4}, indicating that QNG reliably finds high-accuracy solutions regardless of initialization. This consistency stems from the Fisher information matrix's geometric guidance toward the optimal solution.

Momentum shows intermediate performance with median error around 5×10^{-5} and broader distribution ranging from 10^{-5} to 10^{-3}. Several outliers reach 10^{-1}, suggesting that momentum's acceleration can occasionally lead to suboptimal convergence depending on initial conditions.

Adam exhibits a wider distribution with median error approximately 4×10^{-4}, spanning from 3×10^{-5} to 3×10^{-3}. Despite this spread, Adam shows fewer extreme outliers than Momentum, demonstrating reasonable reliability. The adaptive learning rates provide some protection against poor initializations, though not to the extent of QNG's geometric optimization.

Standard GD shows the poorest robustness with the widest distribution. The median error is approximately 2×10^{-3}, with high variance spanning from 2×10^{-5} to 10^{-1}. Multiple outliers appear at 10^{-1} or higher, indicating frequent failure to converge to acceptable solutions. This high sensitivity to initialization makes plain GD unsuitable for production VQE without careful problem-specific tuning.

Quantitatively, using success criterion $\Delta E < 10^{-3}$, the approximate success rates are: QNG $\approx 97\%$ (29/30 runs), Momentum $\approx 73\%$ (22/30), Adam $\approx 50\%$ (15/30), and GD $\approx 23\%$ (7/30). QNG's near-perfect success rate makes it the most reliable choice when solution quality is critical.

Figure 5: Final error distribution over 30 random initializations using parameter shift gradients. Boxplots show median (orange line), interquartile range (box), whiskers (1.5×IQR), and outliers (circles). QNG achieves tightest distribution with 97% success rate, while GD shows widest spread with only 23% success.

Key Finding: QNG offers the best combination of accuracy and robustness, succeeding across nearly all initializations. Adam provides a practical middle ground with 50% success rate and no

geometric overhead. Standard GD's high failure rate (77%) makes it unreliable for production use without problem-specific initialization strategies.

4.4 Step-Size Strategy Impact

Figure 6 compares constant and decaying step sizes for gradient descent (left) and momentum (right) optimizers with parameter shift gradients.

4.4.1 Gradient Descent

For standard GD (left panel), large constant step size ($\eta = 0.10$, orange) converges fastest initially, reaching energy around -2.0 within 20 iterations. However, it stabilizes at higher error ($\sim 3\times10^{-3}$) due to overshooting in the final convergence phase, unable to refine the solution further.

Small constant step size ($\eta = 0.02$, blue) provides stable descent but suffers from extremely slow convergence, requiring over 150 iterations with the trajectory still showing gradual progress at iteration 200. While eventually achieving good accuracy, this rate is impractical for resource-constrained NISQ devices.

The decaying schedule ($\eta_k = 0.10/(1 + 0.05k)$, green) successfully balances these extremes. It starts with the large rate $\eta_0 = 0.10$ for rapid initial progress, then automatically reduces the step size for refinement as optimization proceeds. This adaptive strategy achieves the best combination of speed and final accuracy, converging to near-exact energy (within 10^{-3}) by iteration 50, faster than small constant step while achieving better accuracy than large constant step.

4.4.2 Momentum Optimizer

Momentum exhibits different step-size sensitivity (right panel). All three strategies converge significantly faster than standard GD, with most progress occurring within the first 30 iterations due to velocity accumulation. The large constant step size ($\eta = 0.10$, orange) shows pronounced oscillatory behavior, particularly visible as the curve oscillates around energy -2.0 between iterations 5–25. These oscillations stem from momentum amplifying the overshooting tendency of large step sizes.

Both small constant ($\eta = 0.02$, blue) and decaying schedules (green) provide smooth convergence to the exact ground state. The decaying schedule shows the cleanest trajectory, converging rapidly to $E = -2.2$ by iteration 20 without any oscillations. This demonstrates that adaptive step sizing synergizes particularly well with momentum: the velocity term already provides some adaptive behavior through gradient history, and combining it with a decaying schedule yields optimal stability while maintaining fast convergence.

Key Finding: Step-size strategy significantly impacts both convergence speed and stability. Decaying schedules offer the most robust performance across optimizers by automatically balancing exploration (large initial steps) and exploitation (refined final steps). This adaptive scheduling is particularly beneficial for momentum-based methods, preventing oscillations while preserving acceleration benefits.

4.5 Summary

Our systematic comparison reveals several key insights for VQE optimization:

1. **Gradient estimation**: Finite difference and parameter shift produce nearly equivalent optimization trajectories (difference $< 10^{-5}$), but FD requires 44% fewer circuit evaluations, making it more cost-effective for measurement-limited NISQ implementations.

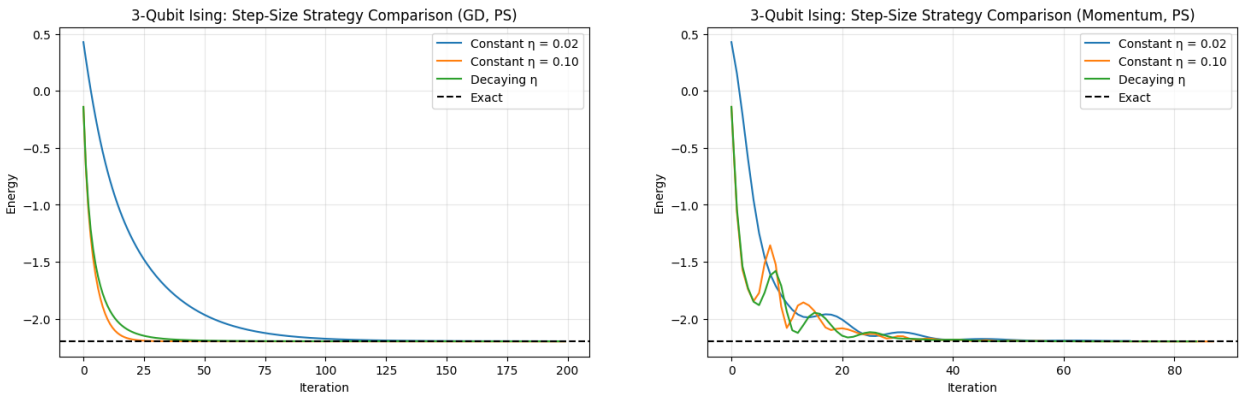

Figure 6: Step-size strategy comparison. Left: Gradient descent showing large constant η overshoots, small constant η converges slowly, and decaying schedule balances speed and accuracy. Right: Momentum with large η oscillates, while decaying schedule provides smoothest, fastest convergence.

2. **Optimizer selection**: QNG achieves fastest convergence and highest accuracy (errors $< 10^{-5}$) with 97% success rate across random initializations. Adam provides a practical alternative with stable performance and no geometric overhead. Standard GD exhibits poor robustness (23% success rate) and should be avoided.

3. **Step-size strategy**: Decaying schedules consistently outperform fixed rates by enabling aggressive initial exploration followed by refined convergence. This is especially critical for momentum-based methods to prevent oscillations while maintaining fast convergence.

4. **Practical implications**: Optimizer choice dominates performance differences, algorithmic design matters far more than gradient estimation precision. For production VQE with limited computational budget, we recommend finite difference gradients with either QNG (when accuracy is critical) or Adam with decaying step size (for balanced performance and robustness).

5 Discussion

Our systematic evaluation reveals nuanced trade-offs between accuracy, computational cost, and robustness that inform practical VQE optimization strategies.

5.1 Optimizer Performance and Practical Trade-offs

QNG's superior performance (errors $< 10^{-5}$, 97% success rate) validates geometric optimization for VQE [10]. By incorporating the quantum Fisher information matrix, QNG accounts for the parameter space geometry, enabling efficient navigation toward optimal solutions. However, this advantage incurs overhead not captured in our noiseless simulations: computing the QFIM requires additional measurements, and matrix inversion scales as $O(d^3)$. For the 9-parameter problem studied here this is negligible, but becomes prohibitive for molecular VQE with 100+ parameters [3].

Adam emerges as a practical compromise. While achieving lower accuracy ($\sim 10^{-2}$ error) than QNG, it requires no geometric computation and provides 50% success rate versus GD's 23%. Its

adaptive per-parameter scaling makes it robust across diverse initialization and problem structures, consistent with findings by Lavrijsen et al. [11] on broader VQE benchmarks.

5.2 Gradient Estimation: Accuracy vs. Cost

A key insight is that gradient estimation method matters less than commonly assumed. Parameter shift provides exact gradients but requires 80% more circuit evaluations than finite difference. Our results show both methods produce nearly identical trajectories (difference $< 10^{-5}$), making FD's 44% cost reduction significant for measurement-limited NISQ devices [5].

However, this conclusion requires caveats. Our FD perturbation ($\epsilon = 10^{-4}$) was tuned for this problem scale. In the presence of shot noise or for ansätze with different parameter sensitivities, FD may become unreliable. Parameter shift's exactness provides insurance for production systems where robustness is paramount [6, 12].

5.3 Step-Size Strategies and Optimizer Synergy

Decaying step-size schedules resolve the exploration-exploitation dilemma inherent in fixed learning rates. They enable aggressive initial progress while ensuring asymptotic refinement, with particularly strong synergy for momentum-based methods. The velocity accumulation in momentum combined with decaying schedules creates complementary adaptive behaviors, while Adam's internal adaptation makes it less sensitive to external scheduling.

5.4 Limitations

Several factors constrain generalizability:

Scale: Our 3-qubit, 9-parameter system is small. Larger molecular problems (50–100 parameters) may exhibit barren plateaus where gradients vanish [4], potentially altering optimizer rankings.

Noise: Statevector simulation isolated optimizer performance from hardware effects. Real devices introduce shot noise, gate errors, and decoherence that may favor inherently robust optimizers like Adam.

Problem diversity: The Ising model's smooth landscape may not reflect molecular electronic structure complexity or frustrated spin systems with multiple local minima.

Ansatz dependence: Hardware-efficient RealAmplitudes may bias results. Problem-inspired ansätze like UCCSD have different parameter correlations [1].

5.5 Practical Recommendations

Based on our findings:

- **High-accuracy applications** (chemistry requiring chemical accuracy): QNG with parameter shift gradients, accepting higher cost for guaranteed quality.
- **Resource-constrained NISQ devices**: Adam with finite difference gradients and decaying step size. Use multiple random restarts to improve success rate.
- **Production systems**: Consider hybrid approaches—Adam for coarse optimization, QNG for refinement. Implement warm-starting when problems change incrementally.

6 Conclusion

Efficient classical optimization is critical to VQE's success on NISQ devices. This work systematically compared gradient estimation methods, optimization algorithms, and step-size strategies, revealing practical trade-offs that inform algorithm selection.

Our key findings: (1) Finite difference and parameter shift produce equivalent trajectories, but FD requires 44% fewer evaluations, making it preferable for resource-constrained devices. (2) QNG achieves superior accuracy ($< 10^{-5}$ error, 97% success) but incurs geometric overhead, while Adam provides robust performance (50% success) without QFIM computation. (3) Decaying step sizes consistently outperform fixed rates across optimizers. (4) Optimizer architecture impacts performance far more than gradient precision.

We recommend Adam with finite difference gradients and decaying step size as a robust default configuration, with QNG reserved for high-accuracy applications. Multiple random restarts significantly improve reliability given initialization sensitivity.

This work demonstrates that classical optimization deserves equal attention to quantum circuit engineering in variational algorithms. As quantum devices scale toward hundreds of qubits, the optimization strategies identified here provide a foundation for tackling larger quantum chemistry and materials science problems in the emerging era of practical quantum computing.

A Implementation Code

This appendix provides the complete implementation of the VQE optimization benchmark used in this study. The code is organized into modular components for reproducibility.

A.1 System Definition: Ising Model

```python
import numpy as np
from qiskit import QuantumCircuit
from qiskit.quantum_info import Operator, Statevector

class IsingSystem:
    """
    Transverse-field Ising model.
    Supports N qubits.
    """
    def __init__(self, J: np.ndarray, h: np.ndarray):
        """
        Parameters
        ----------
        J : (N, N) ndarray
            Ising coupling matrix
        h : (N,) ndarray
            Local fields
        """
        self.N = J.shape[0]
        self.J = J
        self.h = h
        self._build_hamiltonian()
        self._build_initial_state()

    def _build_hamiltonian(self):
        """
        H = - sum_i h_i Z_i - sum_{i<j} J_{ij} Z_i Z_j
```

```
        """
        I = np.eye(2, dtype=complex)
        Z = np.array([[1, 0], [0, -1]], dtype=complex)
        H = np.zeros((2**self.N, 2**self.N), dtype=complex)

        for i in range(self.N):
            ops = [I] * self.N
            ops[i] = Z
            H -= self.h[i] * self._kron(ops)

            for j in range(i + 1, self.N):
                ops = [I] * self.N
                ops[i] = Z
                ops[j] = Z
                H -= self.J[i, j] * self._kron(ops)

        self.hamiltonian = Operator(H)

    def _build_initial_state(self):
        """|0...0> initial state"""
        qc = QuantumCircuit(self.N)
        self.initial_state = Statevector.from_instruction(qc)

    def _kron(self, ops):
        out = ops[0]
        for op in ops[1:]:
            out = np.kron(out, op)
        return out

    def exact_ground_energy(self):
        """Exact solution via diagonalization (benchmark)"""
        eigvals = np.linalg.eigvalsh(self.hamiltonian.data)
        return eigvals[0]
```

Listing 1: Transverse-field Ising model implementation

A.2 Variational Ansatz

```
from qiskit.circuit.library import RealAmplitudes

class HardwareEfficientAnsatz:
    """
    Hardware-efficient RealAmplitudes ansatz.
    Used as the fixed variational ansatz for all experiments.
    """
    def __init__(self, num_qubits: int, reps: int = 1):
        """
        Parameters
        ----------
        num_qubits : int
            Number of qubits
        reps : int
            Number of ansatz layers
        """
        self.num_qubits = num_qubits
        self.reps = reps

        # Build template once
```

```
        self.template = RealAmplitudes(
            num_qubits=num_qubits,
            reps=reps,
            entanglement="full",
            insert_barriers=False
        ).decompose()

        # Total number of variational parameters
        self.num_parameters = self.template.num_parameters

    def circuit(self, theta: np.ndarray) -> QuantumCircuit:
        if len(theta) != self.num_parameters:
            raise ValueError(
                f"Expected {self.num_parameters} parameters, "
                f"got {len(theta)}"
            )
        return self.template.assign_parameters(theta)
```

Listing 2: Hardware-efficient RealAmplitudes ansatz

A.3 Gradient Estimation Methods

```
def finite_difference_gradient(theta, energy_fn, eps=1e-6):
    """
    Finite-difference gradient approximation.
    grad E / grad theta_i ~ [E(theta + eps e_i) - E(theta)] / eps
    """
    grad = np.zeros_like(theta)
    E0 = energy_fn(theta)
    for i in range(len(theta)):
        theta_shift = theta.copy()
        theta_shift[i] += eps
        grad[i] = (energy_fn(theta_shift) - E0) / eps
    return grad

def parameter_shift_gradient(theta, energy_fn, shift=np.pi / 2):
    """
    Exact gradient using the parameter-shift rule.
    grad E / grad theta_i = [E(theta_i + pi/2) - E(theta_i - pi/2)] / 2
    """
    grad = np.zeros_like(theta)
    for i in range(len(theta)):
        theta_plus = theta.copy()
        theta_minus = theta.copy()
        theta_plus[i] += shift
        theta_minus[i] -= shift
        grad[i] = (
            energy_fn(theta_plus) - energy_fn(theta_minus)
        ) / 2.0
    return grad
```

Listing 3: Finite difference and parameter shift gradient implementations

A.4 Step-Size Strategies

```
class StepSize:
    """Base class for step-size strategies."""
```

```
    def __call__(self, k: int) -> float:
        raise NotImplementedError

class ConstantStepSize(StepSize):
    """eta_k = eta"""
    def __init__(self, eta: float):
        self.eta = eta

    def __call__(self, k: int) -> float:
        return self.eta

class DecayingStepSize(StepSize):
    """eta_k = eta0 / (1 + decay * k)"""
    def __init__(self, eta0: float, decay: float):
        self.eta0 = eta0
        self.decay = decay

    def __call__(self, k: int) -> float:
        return self.eta0 / (1.0 + self.decay * k)
```

Listing 4: Step-size scheduling implementations

A.5 Optimization Algorithms

```
class BaseOptimizer:
    def __init__(self, step_size, max_iter=200, tol=1e-6):
        self.step_size = step_size
        self.max_iter = max_iter
        self.tol = tol
        self.history_energy = []
        self.history_params = []

    def step(self, params, grad, k):
        raise NotImplementedError

    def run(self, params0, energy_fn, grad_fn):
        params = params0.copy()
        E_prev = energy_fn(params)

        for k in range(self.max_iter):
            grad = grad_fn(params)
            params = self.step(params, grad, k)
            E = energy_fn(params)

            self.history_energy.append(E)
            self.history_params.append(params.copy())

            if abs(E - E_prev) < self.tol:
                break
            E_prev = E

        return params, E
```

Listing 5: Base optimizer class

```
class GradientDescent(BaseOptimizer):
    def step(self, params, grad, k):
        eta = self.step_size(k)
```

```
        return params - eta * grad
```

Listing 6: Gradient descent optimizer

```
class MomentumGD(BaseOptimizer):
    """
    Heavy-ball / Polyak momentum:
    v_{k+1} = beta * v_k + grad
    theta_{k+1} = theta_k - eta_k * v_{k+1}
    """
    def __init__(self, step_size, beta=0.9, **kwargs):
        super().__init__(step_size=step_size, **kwargs)
        self.beta = beta
        self.v = None

    def step(self, params, grad, k):
        if self.v is None:
            self.v = np.zeros_like(params)
        self.v = self.beta * self.v + grad
        eta = self.step_size(k)
        return params - eta * self.v
```

Listing 7: Momentum optimizer

```
class Adam(BaseOptimizer):
    def __init__(self, lr=0.02, beta1=0.9, beta2=0.999,
                 eps=1e-8, **kwargs):
        super().__init__(step_size=None, **kwargs)
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.eps = eps
        self.m = None
        self.v = None
        self.t = 0

    def step(self, params, grad, k):
        if self.m is None:
            self.m = np.zeros_like(params)
            self.v = np.zeros_like(params)

        self.t += 1
        self.m = self.beta1 * self.m + (1 - self.beta1) * grad
        self.v = self.beta2 * self.v + (1 - self.beta2) * (grad ** 2)

        m_hat = self.m / (1 - self.beta1 ** self.t)
        v_hat = self.v / (1 - self.beta2 ** self.t)

        return params - self.lr * m_hat / (np.sqrt(v_hat) + self.eps)
```

Listing 8: Adam optimizer

```
class QuantumNaturalGradient(BaseOptimizer):
    def __init__(self, step_size, fim_fn, reg=1e-6, **kwargs):
        super().__init__(step_size, **kwargs)
        self.fim_fn = fim_fn
        self.reg = reg

    def step(self, params, grad, k):
        eta = self.step_size(k)
```

```
        F = self.fim_fn(params)
        F_reg = F + self.reg * np.eye(len(params))
        F_inv = np.linalg.pinv(F_reg)
        return params - eta * (F_inv @ grad)
```

Listing 9: Quantum natural gradient optimizer

A.6 Quantum Fisher Information Matrix

```
def fim_fn(theta, ising, ansatz, eps=1e-6):
    """
    Numerical approximation of the quantum Fisher information matrix
    using finite differences of the statevector.
    """
    n = len(theta)
    F = np.zeros((n, n))

    # Reference state
    psi = ising.initial_state.evolve(ansatz.circuit(theta)).data

    # State derivatives
    dpsi = []
    for i in range(n):
        theta_shift = theta.copy()
        theta_shift[i] += eps
        psi_shift = ising.initial_state.evolve(
            ansatz.circuit(theta_shift)
        ).data
        dpsi.append((psi_shift - psi) / eps)
    dpsi = np.array(dpsi)

    # Build QFIM
    for i in range(n):
        for j in range(n):
            F[i, j] = np.real(
                np.vdot(dpsi[i], dpsi[j])
                - np.vdot(dpsi[i], psi) * np.vdot(psi, dpsi[j])
            )

    return F
```

Listing 10: QFIM computation via finite differences

A.7 Experimental Setup

```
# System parameters
N = 3  # Number of qubits

# Ising couplings (1D chain: 0--1--2)
J = np.zeros((N, N))
J[0, 1] = J[1, 0] = 1.0
J[1, 2] = J[2, 1] = 1.0

# Local fields
h = np.array([0.5, -0.5, 0.2])

# Initialize system
```

```
ising = IsingSystem(J, h)

# Initialize ansatz
ansatz = HardwareEfficientAnsatz(num_qubits=N, reps=2)

# Energy function
def energy(theta):
    state = ising.initial_state.evolve(ansatz.circuit(theta))
    return np.real(
        np.vdot(state.data, ising.hamiltonian.data @ state.data)
    )

# Gradient methods
grad_fd = lambda t: finite_difference_gradient(t, energy, eps=1e-4)
grad_ps = lambda t: parameter_shift_gradient(t, energy)

# Exact ground state energy (benchmark)
E_exact = ising.exact_ground_energy()
print(f"Exact ground-state energy: {E_exact}")
print(f"Number of parameters: {ansatz.num_parameters}")
```

Listing 11: Complete experimental configuration

A.8 Running Experiments

```
# Initialize parameters
np.random.seed(42)
theta0 = np.random.uniform(0, 2*np.pi, ansatz.num_parameters)

# Create optimizer
eta = ConstantStepSize(eta=0.05)
optimizer = GradientDescent(step_size=eta, max_iter=200)

# Run optimization
theta_final, E_final = optimizer.run(theta0, energy, grad_ps)

print(f"Final energy: {E_final}")
print(f"Error to exact: {E_final - E_exact}")
print(f"Iterations: {len(optimizer.history_energy)}")
```

Listing 12: Example: Single optimization run

```
def run_many(optimizer_factory, grad_fn, n_trials=30, seed0=0):
    finals = []
    iters = []

    for s in range(seed0, seed0 + n_trials):
        np.random.seed(s)
        theta0 = np.random.uniform(0, 2*np.pi, ansatz.num_parameters)

        opt = optimizer_factory()
        _, _ = opt.run(theta0, energy, grad_fn)

        finals.append(opt.history_energy[-1])
        iters.append(len(opt.history_energy))

    return np.array(finals), np.array(iters)

```

```python
# Example usage
eta = ConstantStepSize(eta=0.05)
gd_factory = lambda: GradientDescent(step_size=eta, max_iter=200)
adam_factory = lambda: Adam(lr=0.02, max_iter=200)

E_gd, it_gd = run_many(gd_factory, grad_ps)
E_adam, it_adam = run_many(adam_factory, grad_ps)

print(f"GD mean error: {np.mean(E_gd - E_exact):.2e}")
print(f"Adam mean error: {np.mean(E_adam - E_exact):.2e}")
```

Listing 13: Example: Robustness analysis over multiple initializations

References

[1] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, p. 4213, 2014.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 10th anniversary ed., 2010.

[3] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, "Quantum computational chemistry," *Reviews of Modern Physics*, vol. 92, no. 1, p. 015003, 2020.

[4] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[5] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[6] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Physical Review A*, vol. 99, no. 3, p. 032331, 2019.

[7] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.

[8] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, "Quantum natural gradient," *Quantum*, vol. 4, p. 269, 2020.

[11] W. Lavrijsen, A. Tudor, J. Müller, C. Iancu, and W. De Jong, "Classical optimizers for noisy intermediate-scale quantum devices," *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 267–277, 2020.

[12] D. Wierichs, C. Gogolin, and M. Kastoryano, "General parameter-shift rules for quantum gradients," *Quantum*, vol. 6, p. 677, 2022.

Table 1: Experimental configuration and hyperparameters for all optimization methods.

Component	**Parameter**	**Value**
Problem Setup		
Hamiltonian	Type	Transverse-field Ising
	Qubits	3
	Couplings J	$J_{01} = J_{12} = 1.0$
	Fields h	$(0.5, -0.5, 0.2)$
	Exact ground energy	-2.2
Ansatz	Type	RealAmplitudes
	Repetitions	2
	Entanglement	Full
	Parameters d	9
	Initial state	$\|000\rangle$
Gradient Methods		
Finite Difference (FD)	Perturbation ϵ	10^{-4}
	Evaluations per gradient	$d + 1 = 10$
Parameter Shift (PS)	Shift angle	$\pi/2$
	Evaluations per gradient	$2d = 18$
Optimization Algorithms		
Gradient Descent (GD)	Step size η	$0.02, 0.05, 0.10$
Momentum	Momentum β	0.9
	Step size η	$0.02, 0.05, 0.10$
Adam	Learning rate η	0.02
	β_1	0.9
	β_2	0.999
	ϵ	10^{-8}
Quantum Natural Gradient	Step size η	0.05
	Regularization λ	10^{-2}
	QFIM approximation ϵ	10^{-6}
Step-Size Strategies		
Constant	Values tested	$\{0.02, 0.05, 0.10\}$
Decaying	Initial η_0	0.10
	Decay rate	0.05
	Schedule	$\eta_k = \eta_0/(1 + 0.05k)$
Experimental Protocol		
Convergence	Maximum iterations	200
	Tolerance	$\|E_{k+1} - E_k\| < 10^{-6}$
Initialization	Distribution	$\mathcal{U}(0, 2\pi)$
	Single-run seed	42
	Robustness trials	30 (seeds 0–29)
Success criterion	Error threshold	$\Delta E < 10^{-3}$
Simulation	Type	Noiseless statevector