An Adaptive Optimizer for Measurement-Frugal Variational Algorithms

Jonas M. Kübler[1,2], Andrew Arrasmith[1], Lukasz Cincio[1], and Patrick J. Coles[1]

[1]Theoretical Division, MS B213, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.
[2]Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, 72076 Tübingen, Germany.

Variational hybrid quantum-classical algorithms (VHQCAs) have the potential to be useful in the era of near-term quantum computing. However, recently there has been concern regarding the number of measurements needed for convergence of VHQCAs. Here, we address this concern by investigating the classical optimizer in VHQCAs. We introduce a novel optimizer called individual Coupled Adaptive Number of Shots (iCANS). This adaptive optimizer frugally selects the number of measurements (i.e., number of shots) both for a given iteration and for a given partial derivative in a stochastic gradient descent. We numerically simulate the performance of iCANS for the variational quantum eigensolver and for variational quantum compiling, with and without noise. In all cases, and especially in the noisy case, iCANS tends to out-perform state-of-the-art optimizers for VHQCAs. We therefore believe this adaptive optimizer will be useful for realistic VHQCA implementations, where the number of measurements is limited.

1 Introduction

There are various strategies to make use of noisy intermediate-scale quantum (NISQ) computers [1]. One particularly promising strategy is to push most of the algorithmic complexity onto a classical computer while running only a small portion of the computation on the NISQ device. This is the idea behind variational hybrid quantum-classical algorithms (VHQCAs) [2]. VHQCAs employ a quantum computer to efficiently estimate a cost function that depends on the parameters of a quantum gate sequence, and then leverage a classical optimizer to minimize this cost. VHQCAs intend to achieve a quantum advantage with NISQ computers by finding short-depth quantum circuits that at least approximately solve some problem. VHQCAs have been proposed for many applications including ground-state preparation, optimization, data compression, simulation, compiling, factoring, diagonalization, and others [3–24].

A concern about VHQCAs is that they might require prohibitively many quantum measurements (shots) in order to achieve convergence of the cost function [25], especially for applications like quantum chemistry that require chemical accuracy [26, 27]. In response to this concern, there has been an recent explosion of papers looking to improve the measurement frugality of VHQCAs by simultaneously measuring commuting subsets of the Pauli operators needed for the cost function [28–34].

Here, we approach the problem from a different direction by aiming to improve the classical optimizer. There have been several recent efforts to improve optimizers for VHQCAs [35–39]. Our approach is different from these works in that the optimizer we propose is specifically constructed to achieve measurement frugality. In particular, we develop an adaptive optimizer that is adaptive in two senses: it frugally adjusts the number of shots for a given iteration and for a given partial derivative.

Our method is inspired by the classical machine learning algorithm named Coupled Adaptive Batch Size (CABS) [40]. For pedagogical reasons, we first directly adapt the CABS algorithm to VHQCA applications and call the resulting algorithm Coupled Adaptive Number of Shots (CANS). In order to achieve greater measurement frugality, we go beyond direct adaptation and modify the optimizer to account for dif-

Patrick J. Coles: pcoles@lanl.gov

ferences in the number of shots needed to estimate individual components of the gradient. We call this method individual-CANS (iCANS).

While iCANS is conceptually simple, it nevertheless performs very well. Using IBM's simulator [41], we implement iCANS and other state-of-the-art optimizers such as Adam [42], SPSA [43], and sequential gate optimization [37, 38] for both the variational quantum eigensolver [3] and variational quantum compiling [14–16, 18]. We find that iCANS on average performs the best. This is especially true for our implementations in the presence of noise, i.e., with IBM's simulator of their NISQ device. This is encouraging since VHQCAs must be able to run in the presence of noise to be practically useful.

Ultimately, one can take a multi-pronged approach to reducing measurements in VHQCAs, e.g., by combining our measurement-frugal classical optimizer with the recent advances on Pauli operator sets in Refs. [28–34]. However, one can apply our optimizer to VHQCAs that do not involve the measurement of Pauli operator sets (e.g., the VHQCAs in [7–9]). In this sense, our work is relevant to all VHQCAs.

In what follows, we first give a detailed review of various optimizers used in the classical machine learning and quantum circuit learning literature. We remark that this lengthy review aims to assist readers who may not have a background in classical optimization, as this article is intended for a quantum-computing audience. (Experienced readers can skip to Section 3.) We then present our adaptive optimizer, followed by the results of our numerical implementations.

2 Background

2.1 Gradient Descent

One standard approach to minimization problems is gradient descent, where the optimizer iteratively steps along the direction in parameter space that is locally "downhill" (i.e., decreasing) for some function $f(\boldsymbol{\theta})$. Mathematically, we can phrase the step at the t-th iteration as

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \boldsymbol{\nabla} f(\boldsymbol{\theta}^{(t)}), \tag{1}$$

where α is called the *learning rate*. If one takes a large learning rate, one cannot be sure that one will not go too far and possibly end up at a higher point. For a small learning rate one is more guaranteed to keep making incremental progress (assuming the change in slope is bounded), but it will take much longer to get to a minimum. Knowing an upper bound on the slope is therefore very helpful in determining the appropriate learning rate.

To formalize this discussion, we review the notion of Lipschitz continuous gradients. The gradient of a function f is Lipschitz continuous if there exists some L (called the Lipschitz constant) such that

$$\|\boldsymbol{\nabla} f(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\nabla} f(\boldsymbol{\theta}^{(t)})\| \leqslant L \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|, \tag{2}$$

for all $\boldsymbol{\theta}^{(t+1)}$ and $\boldsymbol{\theta}^{(t)}$. (We note that in our notation the $\|\cdot\|$ without a subscript denotes the ℓ_2 or Euclidean norm.) When this holds, we can see that the fractional change in the gradient over the course of one step is bounded by αL, meaning that for sufficiently small α we can be sure that we are following the gradient. In fact, the convergence of the basic gradient descent method is guaranteed for deterministic gradient evaluations so long as $\alpha < 2/L$ [40]. In machine learning contexts L is usually unknown, but for VHQCAs it is often possible to determine a good bound. We discuss this alongside an analytic formula for estimating gradients for VHQCAs in the next subsection.

2.2 Gradient Estimation

Working with the exact gradient is often difficult for two reasons. First the gradient can depend on quantities that are expensive to estimate with high precision. Second, it might be that no analytic form for the gradient formula is accessible, and hence the gradient must be approximated by finite differences. In the following we discuss the two scenarios in more detail.

2.2.1 Analytic gradients

If one has sufficient knowledge of the structure of the optimization problem under consideration, it might be possible to find analytic expressions for the gradient of the function. In deep learning this is what is provided via the backpropagation algorithm, which allows one to take analytic derivatives with respect to all parameters [44]. However

these formulas are usually expressed as an average over the full sample one has available in a learning task. To decrease the cost of evaluating the gradient often only a subset of the full sample, a so-called mini-batch, is used to get an unbiased estimate of the gradient [44]. This introduces a trade-off between the cost of the gradient estimation and its achieved precision.

In VHQCAs there exist similar scenarios where it is possible to analytically compute the gradients [45–47]. For example if the parameters describe rotation angles of single-qubit rotations and the cost function is the expectation value of some operator $\boldsymbol{A}$, $f = \langle \boldsymbol{A} \rangle$, partial derivatives can be computed as

$$\partial_{\theta_i} f(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta} + \frac{\pi}{2}\hat{e}_i) - f(\boldsymbol{\theta} - \frac{\pi}{2}\hat{e}_i)}{2}, \qquad (3)$$

i.e., the partial derivative is determined by the value of the cost function if one changes the i-th component by $\pm\pi/2$. However, the value of the cost function can only be estimated from a finite number of measurements, and this number of measurements as well as the noise level of the computation itself determine the precision of the gradient estimates. Therefore it is important to understand how to choose the number of shots, and keep in mind that for VHQCAs the gradient estimate is always noisy to some extent, even though it is referred to as analytical.

An immediate extension of this is that (3) can be used recursively to define higher derivatives. This result then allows one to determine a usefully small upper bound on L in (2). In particular, we note for operators with bounded eigenspectra, the largest magnitude of a derivative of any order we can find with (3) is precisely half the difference between the largest and smallest eigenvalues $\lambda_{\max}$ and $\lambda_{\min}$, respectively. Thus,

$$L \leqslant \frac{\lambda_{\max} - \lambda_{\min}}{2}. \qquad (4)$$

For the common case where the eigenspectrum is unknown but we know how to decompose $\boldsymbol{A}$ into a weighted sum over tensor products of Pauli matrices, $\boldsymbol{A} = \sum_i a_i \boldsymbol{\sigma}_i$, we can bound the highest and lowest eigenvalues in turn by $\sum_i |a_i|$ and $-\sum_i |a_i|$, respectively, which gives

$$L \leqslant \sum_i |a_i|. \qquad (5)$$

By setting equality in (5) (or (4) when we have more information), we therefore find a useful Lipschitz constant.

2.2.2 Finite Differencing

If one does not have access to analytical gradients, one way to approximate the partial derivatives is by taking a finite δ step in parameter space

$$\partial_{\theta_i} f(\boldsymbol{\theta}) \approx \frac{f(\boldsymbol{\theta} + \delta\hat{e}_i) - f(\boldsymbol{\theta} - \delta\hat{e}_i)}{2\delta}. \qquad (6)$$

Again, as in the analytical case, the function values need to be estimated by a finite number of shots introducing statistical noise. However, as opposed to the analytic case, the estimate (6) is systematically wrong, with an error that scales with δ^2. Therefore, one might want to decrease the parameter δ during an optimization procedure using such a gradient estimate. Intuitively this makes the optimization harder, and was recently discussed in the context of VHQCAs [48].

2.3 Noisy Gradient Descent

For the case where one has noise in one's measurement of the gradient, the analysis of a gradient descent procedure becomes more complicated as the best one can achieve are statements about the behavior that can be expected on average. However, so long as one's estimates are unbiased (i.e., repeated estimates average to the true gradient) one should still end up near a minimum. This idea is at the heart of all stochastic gradient descent methods which we discuss now.

2.3.1 Stochastic/Mini-Batch Gradient Descent

In cases such as VHQCAs (as well as some machine learning applications), we cannot access the gradients directly and therefore need to estimate the the gradients by sampling from some distribution. A standard approach to this case is to choose some number of samples that are needed to achieve a desired precision. This method is known as either stochastic or mini-batch gradient descent. (A mini-batch here refers to a collection of samples, usually much smaller than the total population.)

The number of samples as well as the learning rate are usually set heuristically, in order to

balance competing interests of efficiency and precision. First, when collecting samples is computationally expensive, it can sometimes be more efficient to take less accurate gradient estimates in order to converge faster, though doing so can be detrimental if it means that one ends up needing to perform an inordinate number iterations [49]. Second, it does not make sense to attempt to achieve a precision greater than intrinsic accuracy of the distribution from which one samples. If there is some error expected in the representation of the distribution one samples the gradients from, there is therefore an upper bound on the number of samples that it is sensible to take based on that accuracy [49]. For the case of VHQCAs, this often means that the upper limit on the number of samples, $s_{\max}$ depends on the (usually unknown) bias b_{noise} introduced to the gradient measurements by the noise of the physical quantum device:

$$s_{\max} \simeq \frac{\mathrm{Var}(f(\boldsymbol{\theta}))}{b^2_{\text{noise}}(\boldsymbol{\theta}))}. \tag{7}$$

Since for VHQCAs this bias is a function of the unknown, time varying device noise for the specific gate sequence, often the best one can do is to make a rough estimate about its order of magnitude and use that in the denominator.

Typically, the number of samples as well as the learning rate are heuristically adjusted based on the structure of the cost landscape as well as the error level. When little information is known about the optimization problem, the minimization process is optimized either by manual trial and error until an acceptable choice is found or using a hyper-parameter optimization strategy [50].

For a stochastic gradient approach to converge quickly, it is often helpful to decrease the error in the optimization steps during the run of the optimization. This can be done by either decreasing the learning rate α, or minimizing the noise in the gradient estimates. The following two subsections introduce two methods from machine learning that respectively take these two strategies.

2.3.2 Adam

Adam is a variant of stochastic gradient in which the step that is taken along each search direction is adapted based on the first and second moment of the gradient [42]. To do this, one takes an exponential decaying average of the first and second moment (m_t and v_t, respectively) for each component of the gradient individually

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{8}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{9}$$

where the square is understood element-wise, g_t is the gradient estimate at step t, and β_1, β_2 are the constants that determine how slowly the variables are updated. The parameters are then updated with the following rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{10}$$

where $\hat{m}_t$ ($\hat{v}_t$) is an initialization-bias-corrected version of m_t (v_t), and ϵ is a small constant to ensure stability [42]. One particular feature of Adam is that the adaptation happens individually for each component of the gradient. We also briefly mention that there is a recent modification to Adam that looks promising, called Rectified Adam (RAdam) [51]. RAdam essentially selectively turns on the adaptive learning rate once the variance in the estimated gradient becomes small enough.

While Adam has made a large impact in deep learning, to our knowledge it has not been widely considered in the context of VHQCAs.

2.3.3 CABS

Balles et al. analyzed the problem of choosing the sample size in the context of optimizing neural networks by stochastic gradient descent [40]. Their approach is to find the number of samples s that maximizes the expected gain per sample at each iteration.

In the following we denote the i-th component of the estimated gradient by g_i, the empirical variance of the estimate g_i by S_i, the actual gradient by $\boldsymbol{\nabla} f$, and the actual covariance matrix (in the limit of infinite samples or shots) of the gradient estimation by Σ. Balles et al. introduce a lower bound $\mathcal{G}$ on the gain (improvement in the cost function) per iteration. Accounting for the finite sampling error, they find that the average value of $\mathcal{G}$ is [40]

$$\mathbb{E}[\mathcal{G}] = \left(\alpha - \frac{L\alpha^2}{2}\right) \|\boldsymbol{\nabla} f\|^2 - \frac{L\alpha^2}{2s}\mathrm{Tr}(\Sigma). \tag{11}$$

As an immediate consequence, they then find that the expected gain at any step has a positive lower bound if

$$\alpha \leqslant \frac{2\|\boldsymbol{\nabla} f\|^2}{L\left(\|\boldsymbol{\nabla} f\|^2 + \mathrm{Tr}(\Sigma)/s\right)}. \tag{12}$$

By taking a small but fixed α, Balles et al. then maximize the lower bound on the expected gain per sample by taking

$$s = \frac{2L\alpha}{2 - L\alpha} \frac{\mathrm{Tr}(\Sigma)}{\|\boldsymbol{\nabla} f\|^2} \tag{13}$$

samples [40]. Unfortunately, this formula depends on quantities Σ and $\boldsymbol{\nabla} f$ that are not accessible. Therefore in CABS, Σ is replaced by an estimator $\hat{\Sigma}$ and, specializing to the case where the minimum value of f is known to be zero, $\|\boldsymbol{\nabla} f\|^2$ is replaced by f/α as the gradient estimator is biased. Since the Lipschitz constant is also often unknown in the machine learning problems they were considering, they also drop the factor of $2L\alpha/(2 - L\alpha)$ [40]. CABS then proceeds as a stochastic gradient descent with a fixed learning rate and a number of samples that is selected at each iteration based on (13) with the quantities measured at the previous point, making the assumption that the new point will be similar to the old point.

As discussed in the next section, our adaptive optimizer for VHQCAs is built upon the ideas behind CABS (particularly (13)), although our approach differs somewhat.

2.4 SPSA

The simultaneous perturbation stochastic approximation (SPSA) algorithm [43] is explicitly designed for a setting with only noisy evaluation of the cost function, where no analytic formulas for the gradients are available. It is also a descent method, however, instead of estimating the full gradient, a random direction is picked and the slope in this direction is estimated. Based on this estimate a downhill step in the sampled direction is taken:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha_t \boldsymbol{g}(\boldsymbol{\theta}^{(t)}). \tag{14}$$

Here $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$ is the estimated slope in the random direction and estimated as [52]:

$$\boldsymbol{g}(\boldsymbol{\theta}^{(t)}) = \frac{f(\boldsymbol{\theta}^{(t)} + c_t \boldsymbol{\Delta}_t) - f(\boldsymbol{\theta}^{(t)} - c_t \boldsymbol{\Delta}_t)}{2c_t} \boldsymbol{\Delta}_t^{-1}, \tag{15}$$

where $\boldsymbol{\Delta}_t$ is the random direction sampled for the t-th step and $\boldsymbol{\Delta}_t^{-1}$ simply denotes the vector with its element-wise inverses. In order to ensure convergence the finite difference parameter c_t as well as the learning rate α_t have to be decreased over the optimization run. This is commonly done by using a prefixed schedule [52]. In this approach, we have

$$\alpha_t = \frac{\alpha_0}{(1+k)^\beta} \qquad \text{and} \qquad c_t = \frac{c_0}{(1+k)^\gamma}. \tag{16}$$

In the original formulation, the idea is usually to estimate the cost function in (15) by a single measurement. However, in a quantum setting it seems intuitive to take a larger number of measurements for the estimation, as was done in [53].

2.5 Sequential Subspace Search

Another approach to optimizing a multivariate cost function is to break the problem into subparts which are independently easier to handle. The generic idea is to define a sequence of subspaces of parameter space to consider independently. These methods then approach a local minimum by iteratively optimizing the cost function on each subspace in the sequence. Now we discuss two instances of this approach: the famous Powell method [54] as well as a recently proposed method specialized to VHCQAs [37, 38].

2.5.1 Powell Algorithm

The Powell algorithm [54] is a very useful gradient-free optimizer that specializes the subspace search to the case of sequential line searches. Specifically, starting with some input set of search vectors $V = \{\boldsymbol{v}_i\}$ (often the coordinate basis vectors of the parameter space) this method sequentially finds the set of displacements $\{a_i\}$ along each search vector that minimizes the cost function. Next, the method finds the $\boldsymbol{v}_j$ associated with the greatest displacement, $a_j = \max(a_i)$. This $\boldsymbol{v}_j$ is then replaced with the total displacement vector for this iteration, namely:

$$\boldsymbol{v}_j \to \sum_i a_i \boldsymbol{v}_i, \tag{17}$$

and then the next iteration begins with this updated set of search vectors. This replacement scheme accelerates the convergence and prevents

the optimizer from being trapped in a cyclic pattern. In practice, the displacements a_i are typically found using Brent's method [55], but in principle any gradient-free scalar optimizer could work. (Gradient-based scalar optimizers would make Powell's method no longer "gradient-free.")

2.5.2 Sequential Optimization by Function Fitting

In the special case of VHQCAs where the cost function is expressed as an expectation value of some Hermitian operator and the quantum circuit is expressed as fixed two-qubit gates and variable single-qubit rotations, it is possible to determine the functional form of the cost function along a coordinate axis [37]. After fitting a few parameters, it becomes possible to compute where the analytic minimum should be in order to find the optimal displacement along any given search direction. This can be scaled up to finding the analytic minimum (exact up to distortions from noise) on some subspace that is the Cartesian product of coordinate axes, though this is hampered by the fact that the number of parameters that must be fit scales exponentially with the dimension of the subspace [37]. We will refer to this algorithm as the Sequential Optimization by Function Fitting (SOFF) algorithm. We note that a very similar method was published shortly after SOFF [38]. The primary difference was the incorporation of the Anderson and Pulay convergence acceleration procedures used in computational chemistry [56, 57].

We note that, though SOFF and Powell are closely related, due to the limitation to only searching along coordinate axes, it is not possible to take arbitrary search directions, thus SOFF is not quite a special case of Powell's method. For VHQCA problems where it is applicable, SOFF has been demonstrated to be highly competitive with or better than other standard optimization schemes like Powell's method [37, 38].

3 Adaptive Shot Noise optimizer

As mentioned above, the basic idea behind our approach is similar to that of CABS [40], but we implement those ideas in a different way. Specifically, by implementing different estimates for the inaccessible quantities in (13) that are suitable to the number of shots in a VHQCA (rather than the batch size in a machine learning method), we arrive at a variant of CABS we name *Coupled Adaptive Number of Shots* (CANS). Recognizing that a different number of shots might be optimal for estimating each component of the gradient in VHQCAs, we further develop this variation into *individual-CANS* (iCANS), which is our main result. For pedagogical purposes, we first introduce CANS and then present iCANS.

3.1 CANS

We now discuss our adaptation of CABS to the setting of VHQCAs. In order to use the number of shots recommended by the CABS method, we need to rewrite (13) using only quantities that are accessible. Making use of the parameter shift rule (3), we have access to the Lipschitz constant L given by (5). An unbiased estimate of $\mathrm{Tr}(\Sigma)$ is given by $\sum_{i=1}^{d} S_i = \|S\|_1$, i.e., by the empirical variances of the gradient components. (Here and below d is the number of parameters being optimized.) The naive estimate of $\|\boldsymbol{\nabla} f\|^2$ is $\|g\|^2$, with $g := (g_1, ..., g_l)^T$ the estimated gradient. This estimator is biased (see Equation (17) of [40]), however using a bias-corrected version is numerically unstable. With these choices, we then define CANS as the CABS algorithm with (13) replaced by

$$s = \frac{2L\alpha}{2 - L\alpha} \frac{\|S\|_1}{\|\boldsymbol{g}\|^2}. \tag{18}$$

We note that the learning rate α must be less than $2/L$ with this formalism. The CANS algorithm is included in Appendix B for completeness. For the remainder of the paper we will focus on iCANS, which we introduce next.

3.2 iCANS

The CANS algorithm is inspired by CABS [40], which was designed for applications in deep learning. Therein for each data point the full gradient is evaluated, and noise arises by considering only a minibatch of the full sample. In VHCQAs, however, each individual partial derivative is estimated independently. This gives us the freedom to distribute measurements over the estimation of the partial derivatives more effectively. This is the idea behind iCANS, which is shown in Algorithm 1 and described below.

Algorithm 1 Stochastic gradient descent with iCANS1/2. The function $iEvaluate(\boldsymbol{\theta}, \boldsymbol{s})$ evaluates the gradient at $\boldsymbol{\theta}$ using s_i shots for the i-th derivative via the parameter shift rule (3). This function returns the estimated gradient vector $\boldsymbol{g}$ as well as the vector $\boldsymbol{S}$ whose components are the variances of the estimates of the partial derivatives.

Input: Learning rate α, starting point $\boldsymbol{\theta}_0$, min number of shots per estimation $s_{\min}$, number of shots that can be used in total N, Lipschitz constant L, running average constant μ, bias for gradient norm b

initialize: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$, $s_{\text{tot}} \leftarrow 0$, $\boldsymbol{s} \leftarrow (s_{\min}, ..., s_{\min})^T$, $\boldsymbol{\chi}' \leftarrow (0, ..., 0)^T$, $\boldsymbol{\xi}' \leftarrow (0, ..., 0)^T$, $k \leftarrow 0$
while $s_{\text{tot}} < N$ **do**
  $\boldsymbol{g}, \boldsymbol{S} \leftarrow iEvaluate(\boldsymbol{\theta}, \boldsymbol{s})$
  $s_{\text{tot}} \leftarrow s_{\text{tot}} + 2\sum_i s_i$
  $\boldsymbol{\xi}'_\ell \leftarrow \mu \boldsymbol{\xi}'_\ell + (1-\mu)\boldsymbol{S}_\ell$
  $\boldsymbol{\chi}'_\ell \leftarrow \mu \boldsymbol{\chi}'_\ell + (1-\mu)\boldsymbol{g}_\ell$
  $\boldsymbol{\xi}_\ell \leftarrow \boldsymbol{\xi}'_\ell / (1-\mu^{k+1})$
  $\boldsymbol{\chi}_\ell \leftarrow \boldsymbol{\chi}'_\ell / (1-\mu^{k+1})$
  for $i \in [1, ..., d]$ **do**
    if iCANS1 **then**
      $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha \boldsymbol{g}_i$
    else if iCANS2 **then**
      if $\alpha \leqslant \frac{g_i^2}{L\left(g_i^2 + S_i/s_i + b\mu^k\right)}$ **then**
        $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha \boldsymbol{g}_i$
      else
        $\alpha' \leftarrow \frac{g_i^2}{L\left(g_i^2 + S_i/s_i + b\mu^k\right)}$
        $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha' \boldsymbol{g}_i$
      end if
    end if
    $s_i \leftarrow \left\lceil \frac{2L\alpha}{2-L\alpha} \frac{\xi_i}{\chi_i^2 + b\mu^k} \right\rceil$
    $\gamma_i \leftarrow \frac{1}{s_i}\left[\left(\alpha - \frac{L\alpha^2}{2}\right)\chi_i^2 - \frac{L\alpha^2}{2s_i}\xi_i\right]$
  end for
  $s_{\max} \leftarrow s_{\arg\max_i \gamma_i}$
  $\boldsymbol{s} \leftarrow clip(\boldsymbol{s}, s_{\min}, s_{\max})$
  $k \leftarrow k + 1$
end while

iCANS prioritizes the individual partial derivatives rather than the gradient magnitude as in (11). For this purpose, we define $\mathcal{G}_i$ as our lower bound on the gain (i.e., the improvement in the cost function) associated with the change in parameter θ_i for a given optimization step. Furthermore, we define γ_i as the expected gain per shot (i.e., the expectation value of $\mathcal{G}_i$ divided by the number of shots) as follows:

$$\gamma_i := \frac{\mathbb{E}\left[\mathcal{G}_i\right]}{s_i} = \frac{1}{s_i}\left[\left(\alpha - \frac{L\alpha^2}{2}\right) g_i^2 - \frac{L\alpha^2}{2s_i} S_i\right], \tag{19}$$

where s_i is the suggested number of shots for the estimation of the i-th partial derivative. Note that (19) is an adaptation of (11) to our setting.

In analogy with the CANS approach (see (18)), we estimate the number of shots that maximizes (19) with

$$s_i = \frac{2L\alpha}{2 - L\alpha} \frac{S_i}{g_i^2}. \tag{20}$$

As with CANS, we again note that this formalism is only valid if $\alpha < 2/L$. The idea now is to update each parameter with a gradient-descent step, where each partial derivative is estimated with its individual optimal number of shots. However, empirically those parameters that are close to a local optimal value (hence have a small expected gain) require a large number of shots, while parameters that are far from convergence (and hence usually have a large expected gain) require a small number of shots. We therefore restrict our algorithm to not take more shots for any partial derivative than a cap we will call $s_{\max}$. We take $s_{\max}$ to be the number of shots needed in order to estimate the partial derivative for the parameter $\theta_{i_{\max}}$, where $i_{\max}$ is the index associated with highest expected gain per shot. In other words:

$$i_{\max} = \arg\max_i \left(\gamma_i\right), \tag{21}$$

$$s_{\max} = s_{i_{\max}}, \tag{22}$$

and we impose $s_i \leqslant s_{\max}$ for all partial derivatives.

We note that the introduction of this cap on the number of shots is a heuristic choice which we find to often be beneficial to shot frugality, but which removes the guarantee that γ_i will be maximized or even positive. In order to preserve this

frugality while retaining the guarantee of positive expected gains, one can also introduce a step that verifies that the learning rate to be used is appropriate after the measurements are taken and adapts it if it is not. Motivated by (12), we check the following condition for each component of the gradient:

$$\alpha \leqslant \frac{g_i^2}{L\left(g_i^2 + S_i/s_i\right)}. \tag{23}$$

When this condition fails to hold for the i-th partial derivative, we temporarily replace α with the right hand side of (23) for the update along that direction. Adding in this check results in a more conservative procedure as it takes smaller steps when needed in order to enforce that $\gamma_i > 0$, and thus restores the guarantee that $\mathbb{E}[\mathcal{G}] > 0$. Below, we will refer to iCANS without this learning rate check as iCANS1 and with it as iCANS2. The distinction between iCANS1 and iCANS2 is made in Algorithm 1 with the conditional statements on lines 10 and 12.

Beyond the core components of iCANS given above, both implementations of iCANS also take two more hyperparameters for increased stability. Since iCANS is intended to be deployed on highly noisy problems, we find that it is beneficial to use smoothed quantities for the gradient and variance when estimating γ_i and s_i. For this reason, we use bias-corrected exponential moving averages χ_i and ξ_i in place of g_i and S_i, respectively, when implementing equations (19) and (20). These exponential moving averages introduce a new parameter, μ, which controls the degree of smoothing and is bounded between 0 and 1. Since the update step is independent of this smoothing, we often find it beneficial to choose μ close to 1 to achieve a steady progression of s_i's. Finally, we also add a regularizing parameter b to the denominators of lines 13, 16, and 20 of Algorthim 1 for numerical stability. By multiplying b by μ^k and choosing b to be small, the bias from this regularizing parameter begins small and exponentially decays as the algorithm progresses.

4 Implementations

In order to compare the performance of iCANS1 and iCANS2 to established methods, we consider two optimization tasks: variational quantum compiling with a fixed input state [14–16, 18] and a variational quantum eigensolver (VQE) [3] for a Heisenberg spin chain.

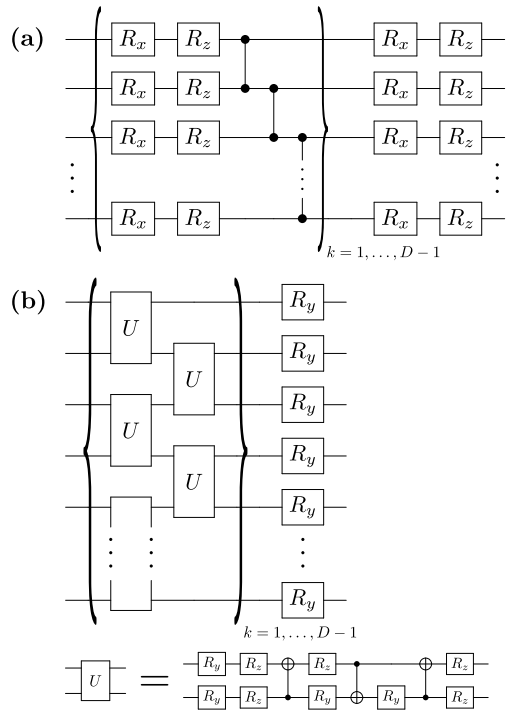

Figure 1: The quantum circuit diagram for the ansatzes we used to construct the unitary operators $U(\boldsymbol{\theta})$ in our implementations. The angles in each rotation gate (denoted as R_j, where j denotes the axis being rotated about) are varied independently. Panel **a** shows the ansatz used in the compiling and Heisenberg spin chain VQE task, and we note that this is the same ansatz used in Ref. [37]. Panel **b** shows the ansatz used when doing the size scaling comparison with the Ising spin chain VQE task.

For our experiments we set the iCANS hyperparameters as $\alpha = 0.1$, $\mu = 0.99$, and $b = 10^{-6}$, except for the case of the system size scaling comparison. For that case, since the Lipschitz constant L grows linearly with the system size, leaving $\alpha = 0.1$ leads to $\alpha > 2/L$ for larger systems, which is invalid for iCANS. We therefore chose $\alpha = 1/L$ for the different length Ising spin chains we consider below.

For the other algorithms we compare to, we will denote the number of shots per operator measurement as s. We will denote algorithm A with s shots per operator measurement as A-s (e.g., SOFF with $s = 1000$ is denoted SOFF-1000). We also note that in the figures and tables below we show the analytical cost and energies that one could achieve with the parameters that the optimizers output, i.e., without hardware noise or shot noise. The optimizers did have to contend with finite statistics and, where indicated,

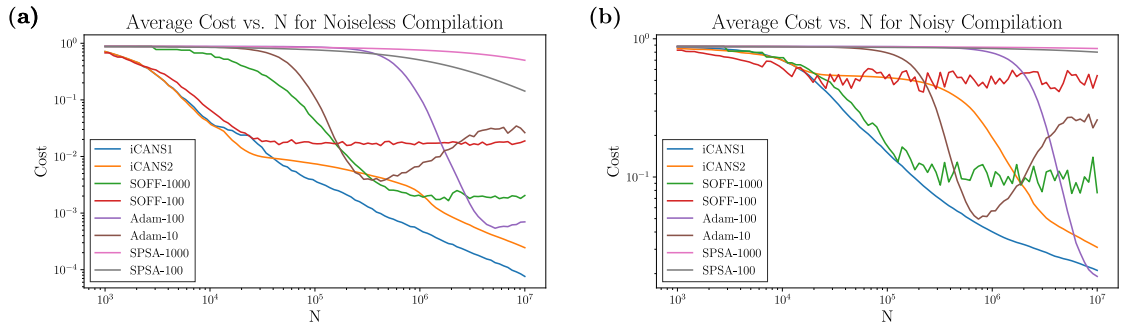

Figure 2: Comparison of performance for the compilation task across one hundred random target states and initial starts. As mentioned in the text, we denote algorithm A with s shots per operator measurement as A-s. Panels **a** and **b** show the average cost value attained as a function of the total number of shots (N) expended for the noiseless and noisy cases, respectively.

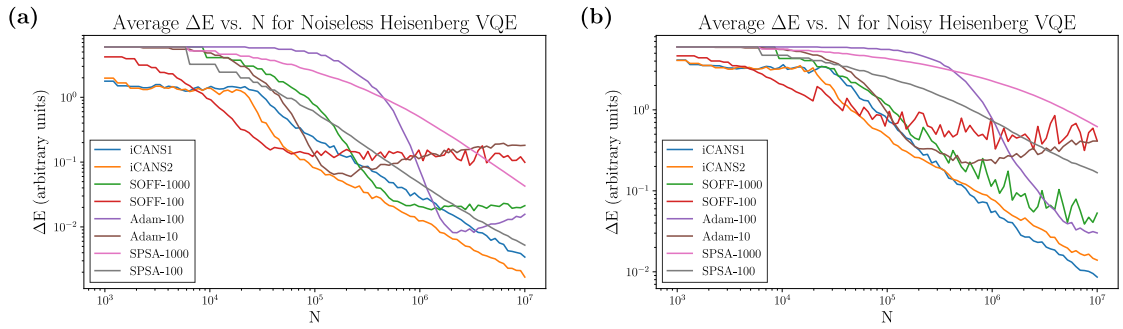

Figure 3: Comparison of performance for the Heisenberg spin chain VQE task across one hundred random starts. Panels **a** and **b** show the average ΔE value (i.e. energy above the ground state energy) attained as a function of the total number of shots (N) expended for the noiseless and noisy cases, respectively.

hardware noise to find those parameters.

In addition to the fixed number of shots they use, the other algorithms we compare to also come with other hyperparameters, which were chosen empirically in an attempt to get the best performance from each. For Adam we used a learning rate of $\alpha = 0.1$ along with the momentum parameter values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For SPSA, we found that the default parameters were the best among those that we tried, and thus we set A to be a tenth of the total number of allowed iterations, $\beta = 0.602$, and $\gamma = 0.101$.

4.1 Variational Compiling with a Fixed Input State

For our first optimization task, we follow [37] and consider as a benchmark the optimization of the following cost function:

$$C = 1 - \left| \langle \mathbf{0} | U(\boldsymbol{\theta}^*)^\dagger U(\boldsymbol{\theta}) | \mathbf{0} \rangle \right|^2 \qquad (24)$$

where $\boldsymbol{\theta}^*$ is a vector of fixed, randomly selected angles and $\boldsymbol{\theta}$ is the vector of angles to be optimized over. For this problem, we construct the parametrized unitary operator $U(\boldsymbol{\theta})$ with the ansatz described in Fig. 1(a), setting $n = 3$ qubits and $D = 6$. (As adding depth and thus more parameters increases the difficulty of the optimization task and amplifies the effect of the noise model, $D = 6$ was chosen to increase the difficulty of the task although shorter depth ansatzes would work here.) We then simulate the optimization procedure with one hundred different random seeds (each of which generates a unique random $\boldsymbol{\theta}^*$ and initial point) and a collection of different optimizers. The results for both the case of a noiseless simulator and the case of a simulator using the noise profile of IBM's Melbourne processor [58] are shown in Fig. 2. For the latter, we emphasize that this noise profile reflects the properties of real, currently available quantum hardware. In addition, the average costs obtained for each optimizer are listed in Tables 1 and 2 with the best value found for each total number of shots expended N shown in bold. Furthermore, see Appendix C for the cumulative probability distributions over cost values, which provide more information than the average cost

Table 1: Noiseless Compilation Average Cost Values

# Shots (N)	10^3	10^4	10^5	10^6	10^7
iCANS1	0.7140	0.0410	**0.0037**	**0.0005**	**0.0001**
iCANS2	0.7149	**0.0386**	0.0074	0.0022	0.0002
SOFF-1000	X	0.6483	0.0430	0.0020	0.0021
SOFF-100	**0.6761**	0.0652	0.0185	0.0162	0.0164
Adam-100	0.8814	0.8807	0.8578	0.1113	0.0007
Adam-10	0.8807	0.8576	0.1108	0.0072	0.0289
SPSA-1000	X	0.8693	0.8426	0.7587	0.5009
SPSA-100	0.8719	0.8455	0.7625	0.5077	0.1428

Table 2: Noisy Compilation Average Cost Values

# Shots (N)	10^3	10^4	10^5	10^6	10^7
iCANS1	0.8711	0.6984	**0.1498**	**0.0402**	**0.0211**
iCANS2	0.8527	0.6982	0.5236	0.1926	0.0310
SOFF-1000	X	0.7302	0.1634	0.1157	0.0912
SOFF-100	**0.8272**	**0.6109**	0.5506	0.4337	0.5740
Adam-100	0.8814	0.8813	0.8791	0.7911	0.0191
Adam-10	0.8813	0.8790	0.7918	0.0556	0.2583
SPSA-1000	X	0.8775	0.8744	0.8679	0.8504
SPSA-100	0.8761	0.8732	0.8669	0.8503	0.8006

Table 3: Noiseless VQE Average Δ Energies

# Shots (N)	10^3	10^4	10^5	10^6	10^7
iCANS1	**1.7732**	1.3746	0.2478	0.0290	0.0034
iCANS2	1.9755	1.3813	**0.0831**	**0.0124**	**0.0017**
SOFF-1000	X	4.0944	0.7970	0.0207	0.0213
SOFF-100	4.2024	**0.9666**	0.1221	0.1536	0.0993
Adam-100	5.9849	5.9849	4.8078	0.0818	0.0157
Adam-10	5.9849	4.8293	0.1431	0.1126	0.1816
SPSA-1000	X	5.1937	2.5710	0.5067	0.0426
SPSA-100	5.9849	3.2301	0.6240	0.0485	0.0052

Table 4: Noisy VQE Average Δ Energies

# Shots (N)	10^3	10^4	10^5	10^6	10^7
iCANS1	4.1202	3.3035	0.8363	**0.0540**	**0.0086**
iCANS2	**4.0782**	3.1897	**0.4918**	0.0796	0.0139
SOFF-1000	X	4.3083	1.1973	0.1137	0.0531
SOFF-100	4.6518	**2.1337**	0.6384	0.7574	0.4090
Adam-100	5.9382	5.9382	5.5313	0.8062	0.0301
Adam-10	5.9849	5.5759	1.0113	0.2428	0.4119
SPSA-1000	X	5.6286	4.3573	2.2965	0.6206
SPSA-100	5.9849	4.7220	2.5740	0.7463	0.1674

values.

4.2 VQE

For our second optimization task, we follow [53] in considering the Heisenberg spin chain with wrapped boundary conditions and the Hamiltonian:

$$H = J \sum_{<ij>} (X_i X_j + Y_i Y_j + Z_i Z_j) + B \sum_i Z_i, \tag{25}$$

where the $<>$ bracket denotes nearest-neighbor pairs. For the purpose of our comparison, we fix $J = 1$ and $B = 3$ and again consider the ansatz described in Fig. 1(a). Running the comparison with $n = 3$ qubits in a triangle and $D = 6$ for the ansatz, we simulate running VQE with one hundred different random seeds and initial points, along with the same set of optimizers as in the benchmark case above. As before, the results for the both a noiseless and a noisy simulator (also using the IBM Melbourne processor's noise profile [58]) are shown in Fig. 3. Again, the average energies obtained for each optimizer are listed in Tables 3 and 4 with the best value found for each total number of shots expended N shown in bold. In addition, see Appendix C for the cumulative probability distributions over energy values, which provide more information than the average energy values.

4.3 Comparison of Scaling

In order to compare the performance of iCANS to that of other optimizers when one scales up the number of qubits, we now consider VQE applied to Ising spin chains of differing lengths with open boundary conditions and the Hamiltonian:

$$H = -\sum_{<ij>} Z_i Z_j - g \sum_i X_i, \tag{26}$$

where the $<>$ bracket again denotes nearest-neighbor pairs. In order to generate enough entanglement in the ground state to require a modest depth, we choose $g = 1.5$ so that we are near but not at the critical point $g = 1$. For this problem, we used the ansatz shown in Fig. 1(b) with $D = 3$ (two repetitions of the block shown in braces), as its performance for this problem was significantly better than the simple ansatz in Fig. 1(a).

The results for a noiseless simulator for 4, 6, 8, 10, and 12 qubit Ising spin chains are shown in Fig. 4.

5 Discussion

Here we report on the behavior of the various optimizers we studied. First we consider SOFF, which is the only optimizer studied here other than iCANS that was formulated specifically for VHQCAs. By leveraging analytical knowledge about the optimization landscape, SOFF's

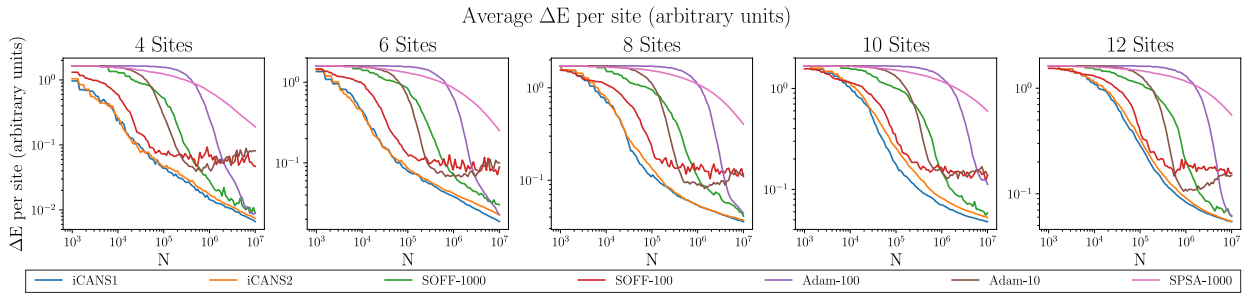

Figure 4: Comparison of performance for the Ising VQE task with different numbers of sites (i.e., qubits) without hardware noise. Each panel shows the average ΔE per site value attained as a function of the total number of shots expended (N) for each number of qubits. Each curve represents the average over ten random starts.

gradient-free method of making single parameter updates allows it to quickly train in low noise environments. However, the limit of the precision when fitting the analytical function with a finite number of shots means that SOFF hits a precision floor and cannot improve past that point. Additionally, hardware noise tends to distort the landscape in such a way that the analytical form no longer provides as good of a fit, making SOFF struggle more relative to the other optimizers considered. In the optimization tasks we looked at here, we found that SOFF was often competitive with iCANS shortly before hitting its precision floor, with SOFF-100 sometimes doing better for a brief interval early on. For example, SOFF-100 was the best performing optimizer for the compilation task with $N = 10^3$ (noiseless and noisy) and $N = 10^4$ (noisy only), as well as for the Heisenberg VQE with $N = 10^4$ (noiseless and noisy).

Adam was originally conceived in the context of machine learning and excels at optimizing in noisy environments. However, in our numerical studies we found that Adam suffered from an instability given the hyperparameters we chose and the number of shots we allowed at each partial derivative evaluation. This instability appears to enter later in the optimization when we are working with more shots, and it can be seen in the upturn of the curves in Figs. 2–4. For the case of the noisy compilation task, Adam-100 looks like it might just be reaching that instability at the end of the allowed shot budget, and slightly outperformed iCANS1 to be the best on average. We note that, similar to what was seen with SOFF, Adam was usually competitive with the iCANS methods before it reached the point where it stopped improving.

Unlike SOFF and Adam, SPSA did not seem to hit a point at which it stopped improving with shot budget, for the chosen hyperparameters. We note though that SPSA is the most sensitive to perturbations of the hyperparameters among the methods studied here and can become very unstable if they are incorrectly chosen. However, if one hits upon the correct hyperparameters, SPSA can be very effective. While for our cases we did not find SPSA outperforming iCANS, we note that for the noiseless Heisenberg VQE task, SPSA-100 was the most competitive with iCANS.

Overall, we find that iCANS performed well on all optimization tasks considered, with either iCANS1 or iCANS2 usually providing the best result for a given total shot budget N. Even when scaling up the system size in the Ising model VQE task (see Fig. 4), we found that iCANS continued to outperform the other optimizers studied. We also note that empirically iCANS1 usually outperformed iCANS2. While iCANS2 provides a benefit by reducing the sensitivity to the input learning rate, so long as the learning rate is chosen well we expect that iCANS1 may tend to perform better.

We remark that while we do not report full results for RAdam [51], we found with preliminary results that it did not seem to provide a substantial improvement over the simpler Adam algorithm for our use cases. Similarly, we found that SOFF with the Anderson acceleration step proposed in [38] did not noticeably improve upon the performance of basic SOFF, and therefore the curves for this method are not shown.

We finally remark about the different performance for the various fixed-shot optimizers with different numbers of shots (e.g., Adam-10 versus Adam-100). This performance difference can be understood as a trade-off between reducing the statistical uncertainty and achieving more itera-

tions before hitting the limit on the total number of shots. When few shots are used, many more iterations might be allowed but the update steps are much noisier, usually meaning that the optimizer can perform more quickly early on but then potentially hits an effective floor due to the precision. Increasing the number of shots will allow more precise updates and thus lowers the precision floor (if present) but means that far fewer iterations can be performed. This is the idea at the heart of iCANS. iCANS uses few shots early on and so achieves a period of noisy but fast descent, but then slows down and computes with greater and greater precision to continue making progress. This strategy allows for shot frugality as well as in principle removing such a precision floor for iCANS.

6 Conclusions

In order to bring about the promise of VHQCAs solving usefully large and complex problems on NISQ devices, one needs a way to perform the requisite optimizations efficiently. As the rate-limiting step of these optimizations will likely be the number of times one must prepare and measure quantum states, it will be important to have optimizers that are frugal in the number of times physical measurements must be performed on a quantum computer.

In this work we introduced two versions of a measurement-frugal, noise-resilient optimizer tailored for VHQCAs. Both of the strategies we propose, iCANS1 and iCANS2, address measurement frugality by dynamically determining the number of measurements needed for each partial derivative of each step in a gradient descent. iCANS1 is the more aggressive version, always taking the same learning rate, while iCANS2 is more cautious and limits the learning rate for steps so that the expected gain is always guaranteed to be positive. Our numerical results indicate that these optimizers may perform comparably or better than other state-of-the-art optimizers. The performance compares especially well in the presence of realistic hardware noise.

iCANS has already found use in the very recent VHQCA literature [18]. Furthermore, after our article was originally posted, a related study of stochastic gradient descent for VHQCAs found that small shot counts can provide rapid improvement in early stages of training [59], which provides further motivation for iCANS.

One potential direction for future work is exploring the possibility of extending our frugal adaptive approach to non-gradient methods, such as SPSA.

Acknowledgements

JMK acknowledges support from the U.S. Department of Energy (DOE) through a quantum computing program sponsored by the Los Alamos National Laboratory (LANL) Information Science & Technology Institute. AA, LC, and PJC acknowledge support from the LDRD program at LANL. PJC also acknowledges support from the LANL ASC Beyond Moore's Law project. This work was also supported by the U.S. DOE, Office of Science, Office of Advanced Scientific Computing Research, under the Quantum Computing Application Teams program.

References

[1] J. Preskill, *Quantum computing in the NISQ era and beyond,* Quantum **2**, 79 (2018).

[2] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, *The theory of variational hybrid quantum-classical algorithms,* New Journal of Physics **18**, 023023 (2016).

[3] 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 **5**, 4213 (2014).

[4] E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm,* arXiv:1411.4028 (2014).

[5] P. D. Johnson, J. Romero, J. Olson, Y. Cao, and A. Aspuru-Guzik, *QVECTOR: an algorithm for device-tailored quantum error correction,* arXiv:1711.02249 (2017).

[6] J. Romero, J. P. Olson, and A. Aspuru-Guzik, *Quantum autoencoders for efficient compression of quantum data,* Quantum Science and Technology **2**, 045001 (2017).

[7] R. LaRose, A. Tikku, É. O'Neel-Judy, L. Cincio, and P. J. Coles, *Variational quantum state diagonalization,* npj Quantum Information **5**, 57 (2019).

[8] A. Arrasmith, L. Cincio, A. T. Sornborger, W. H. Zurek, and P. J. Coles, *Variational consistent histories as a hybrid algorithm for quantum foundations,* Nature Communications **10**, 3438 (2019).

[9] M. Cerezo, A. Poremba, L. Cincio, and P. J. Coles, *Variational quantum fidelity estimation,* Quantum **4**, 248 (2020).

[10] T. Jones, S. Endo, S. McArdle, X. Yuan, and S. C. Benjamin, *Variational quantum algorithms for discovering hamiltonian spectra,* Physical Review A **99**, 062304 (2019).

[11] X. Yuan, S. Endo, Q. Zhao, Y. Li, and S. C. Benjamin, *Theory of variational quantum simulation,* Quantum **3**, 191 (2019).

[12] Y. Li and S. C. Benjamin, *Efficient variational quantum simulator incorporating active error minimization,* Physical Review X **7**, 021050 (2017).

[13] C. Kokail, C. Maier, R. van Bijnen, T. Brydges, M. Joshi, P. Jurcevic, C. Muschik, P. Silvi, R. Blatt, C. Roos, *et al.*, *Self-verifying variational quantum simulation of lattice models,* Nature **569**, 355 (2019).

[14] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, *Quantum-assisted quantum compiling,* Quantum **3**, 140 (2019).

[15] T. Jones and S. C. Benjamin, *Quantum compilation and circuit optimisation via energy dissipation,* arXiv:1811.03147 (2018).

[16] K. Heya, Y. Suzuki, Y. Nakamura, and K. Fujii, *Variational quantum gate optimization,* arXiv:1810.12745 (2018).

[17] S. Endo, Y. Li, S. Benjamin, and X. Yuan, *Variational quantum simulation of general processes,* arXiv:1812.08778 (2018).

[18] K. Sharma, S. Khatri, M. Cerezo, and P. Coles, *Noise resilience of variational quantum compiling,* New Journal of Physics (2020), 10.1088/1367-2630/ab784c.

[19] J. Carolan, M. Mosheni, J. P. Olson, M. Prabhu, C. Chen, D. Bunandar, N. C. Harris, F. N. Wong, M. Hochberg, S. Lloyd, *et al.*, *Variational quantum unsampling on a quantum photonic processor,* arXiv:1904.10463 (2019).

[20] N. Yoshioka, Y. O. Nakagawa, K. Mitarai, and K. Fujii, *Variational quantum algorithm for non-equilirium steady states,* arXiv:1908.09836 (2019).

[21] C. Bravo-Prieto, LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, *Variational quantum linear solver: A hybrid algorithm for linear systems,* arXiv:1909.05820 (2019).

[22] X. Xu, J. Sun, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, *Variational algorithms for linear algebra,* arXiv:1909.03898 (2019).

[23] S. McArdle, T. Jones, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, *Variational ansatz-based quantum simulation of imaginary time evolution,* npj Quantum Information **5**, 1 (2019).

[24] C. Cirstoiu, Z. Holmes, J. Iosue, L. Cincio, P. J. Coles, and A. Sornborger, *Variational fast forwarding for quantum simulation beyond the coherence time,* arXiv:1910.04292 (2019).

[25] D. Wecker, M. B. Hastings, and M. Troyer, *Progress towards practical quantum variational algorithms,* Phys. Rev. A **92**, 042303 (2015).

[26] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya, *et al.*, *Quantum chemistry in the age of quantum computing,* Chemical reviews (2018), 10.1021/acs.chemrev.8b00803.

[27] S. McArdle, S. Endo, A. Aspuru-Guzik, S. Benjamin, and X. Yuan, *Quantum computational chemistry,* arXiv:1808.10402 (2018).

[28] A. Jena, S. Genin, and M. Mosca, *Pauli partitioning with respect to gate sets,* arXiv:1907.07859 (2019).

[29] A. F. Izmaylov, T.-C. Yen, R. A. Lang, and V. Verteletskyi, *Unitary partitioning approach to the measurement problem in the variational quantum eigensolver method,* arXiv:1907.09040 (2019).

[30] T.-C. Yen, V. Verteletskyi, and A. F. Izmaylov, *Measuring all compatible operators in one series of a single-qubit measurements using unitary transformations,* arXiv:1907.09386 (2019).

[31] P. Gokhale, O. Angiuli, Y. Ding, K. Gui, T. Tomesh, M. Suchara, M. Martonosi, and F. T. Chong, *Minimizing state preparations in variational quantum eigensolver by partitioning into commuting families,* arXiv:1907.13623 (2019).

[32] O. Crawford, B. van Straaten, D. Wang,

T. Parks, E. Campbell, and S. Brierley, *Efficient quantum measurement of pauli operators,* arXiv:1908.06942 (2019).

[33] P. Gokhale and F. T. Chong, $o(n^3)$ *measurement cost for variational quantum eigensolver on molecular hamiltonians,* arXiv:1908.11857 (2019).

[34] W. J. Huggins, J. McClean, N. Rubin, Z. Jiang, N. Wiebe, K. B. Whaley, and R. Babbush, *Efficient and noise resilient measurements for quantum chemistry on near-term quantum computers,* arXiv:1907.13117 (2019).

[35] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, *Learning to learn with quantum neural networks via classical neural networks,* arXiv:1907.05415 (2019).

[36] M. Wilson, S. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. Rieffel, *Optimizing quantum heuristics with meta-learning,* arXiv:1908.03185 (2019).

[37] K. M. Nakanishi, K. Fujii, and S. Todo, *Sequential minimal optimization for quantum-classical hybrid algorithms,* arXiv:1903.12166 (2019).

[38] R. M. Parrish, J. T. Iosue, A. Ozaeta, and P. L. McMahon, *A Jacobi diagonalization and Anderson acceleration algorithm for variational quantum algorithm parameter optimization,* arXiv:1904.03206 (2019).

[39] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, *Quantum natural gradient,* arXiv:1909.02108 (2019).

[40] L. Balles, J. Romero, and P. Hennig, in *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI)* (2017) pp. 410–419.

[41] G. A. et.al., Qiskit: An Open-source Framework for Quantum Computing, (2019).

[42] D. P. Kingma and J. Ba, in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)* (2015).

[43] J. C. Spall, *Multivariate stochastic approximation using a simultaneous perturbation gradient approximation,* IEEE transactions on automatic control **37**, 332 (1992).

[44] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning,* Nature **521**, 436 (2015).

[45] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, *Quantum circuit learning,* Phys. Rev. A **98**, 032309 (2018).

[46] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, *Evaluating analytic gradients on quantum hardware,* Phys. Rev. A **99**, 032331 (2019).

[47] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, and N. Killoran, *Pennylane: Automatic differentiation of hybrid quantum-classical computations,* arXiv:1811.04968 (2018).

[48] A. Harrow and J. Napp, *Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms,* arXiv:1901.05374 (2019).

[49] G. G. Guerreschi and M. Smelyanskiy, *Practical optimization for hybrid quantum-classical algorithms,* arXiv:1701.01450 (2017).

[50] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, in *Advances in Neural Information Processing Systems 24* (2011) pp. 2546–2554.

[51] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, *On the variance of the adaptive learning rate and beyond,* arXiv:1908.03265 (2019).

[52] J. C. Spall, *Implementation of the simultaneous perturbation algorithm for stochastic optimization,* IEEE Transactions on aerospace and electronic systems **34**, 817 (1998).

[53] 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 **549**, 242 (2017).

[54] M. J. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivatives,* The Computer Journal **7**, 155 (1964).

[55] R. P. Brent, *Algorithms for Minimization Without Derivatives* (Dover Publications, 2013).

[56] D. G. Anderson, *Iterative procedures for nonlinear integral equations,* Journal of the ACM (JACM) **12**, 547 (1965).

[57] P. Pulay, *Improved scf convergence acceleration,* Journal of Computational Chemistry **3**, 556 (1982).

[58] IBM Q 16 Melbourne backend specification, `https://github.com/Qiskit/`

ibmq-device-information/tree/master/backends/melbourne/V1 (2018).

[59] R. Sweke, F. Wilde, J. Meyer, M. Schuld, P. K. Fährmann, B. Meynard-Piganeau, and J. Eisert, *Stochastic gradient descent for hybrid quantum-classical optimization*, arXiv:1910.01155 (2019).

A The Expected Lower Bound on the Gain per Shot

Here we repeat the derivation provided by [40] for the lower bound on the expected gain per shot (given in (11)), and extend it to our expression lower bounding the expected gain per shot per partial derivative (19).

Assuming that the cost function is admits a Taylor series representation about the current point in parameter space, to quadratic order we have

$$f(\boldsymbol{\theta}') = f(\boldsymbol{\theta}) + \sum_{i=1}^{d}(\theta_i' - \theta_i)\partial_i f(\boldsymbol{\theta}) + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}(\theta_i' - \theta_i)(\theta_j' - \theta_j)\partial_i\partial_j f(\boldsymbol{\theta}). \quad (27)$$

In this way, we approximate the gain (the change in the cost function) we expect after the update step, with $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha\boldsymbol{g}$:

$$f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}') = \alpha\sum_{i=1}^{d} g_i\partial_i f(\boldsymbol{\theta}) - \frac{\alpha^2}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} g_i g_j\partial_i\partial_j f(\boldsymbol{\theta}). \quad (28)$$

If the gradients are Lipschitz continuous, we can achieve a lower bound $\mathcal{G}$ on this quantity using the Lipschitz constant L:

$$\mathcal{G} = \alpha\boldsymbol{\nabla} f(\boldsymbol{\theta})\cdot\boldsymbol{g} - \frac{\alpha^2 L}{2}\|\boldsymbol{g}\|^2. \quad (29)$$

Next we assume that the gradient estimates $\boldsymbol{g}$ have mean $\mathbb{E}[\boldsymbol{g}] = \boldsymbol{\nabla} f(\boldsymbol{\theta})$ and covariance Σ/s, where s is the number of shots used in the estimate. We then have

$$\mathbb{E}\left[\|\boldsymbol{g}\|^2\right] = \sum_{i=1}^{d}\mathbb{E}\left[g_i^2\right] = \|\boldsymbol{\nabla} f(\boldsymbol{\theta})\|^2 + \sum_{i=1}^{d}\Sigma_{ii}/s. \quad (30)$$

Plugging this back into (29) then gives us

$$\mathbb{E}[\mathcal{G}] = \left(\alpha - \frac{L\alpha^2}{2}\right)\|\boldsymbol{\nabla} f\|^2 - \frac{L\alpha^2}{2s}\mathrm{Tr}(\Sigma), \quad (31)$$

which is (11). Dividing both sides by s then gives the expected lower bound on the gain per shot. In order to arrive at (19), we rewrite this expression as:

$$\mathbb{E}[\mathcal{G}] = \sum_{i=1}^{d}\left[\left(\alpha - \frac{L\alpha^2}{2}\right)(\partial_i f)^2 - \frac{L\alpha^2}{2s_i}\Sigma_{ii}\right] \quad (32)$$
$$= \sum_{i=1}^{d}\mathbb{E}[\mathcal{G}_i]$$

Finally, defining $\gamma_i = \mathbb{E}[\mathcal{G}_i]/s_i$ and replacing $\partial_i f$ and Σ_{ii} with their estimators g_i and S_i, respectively, gives (19).

B CANS Algorithm

For the interested reader, we present the algorithm for CANS (Coupled Adaptive Number of Shots) in Algorithm 2, which is an adaptation of the CABS algorithm [40] to the VHQCA setting.

C Cumulative probability distributions for 3-qubit implementations

Here we show the cumulative distribution plots of the cost values or energies achieved by the optimizers we studied for the compilation task (Fig. 5) and the Heisenberg spin chain VQE task (Fig. 6) for various shot budgets.

Algorithm 2 Stochastic gradient descent with CANS. The function $Evaluate(\boldsymbol{\theta}, s)$ evaluates the gradient at $\boldsymbol{\theta}$ using s measurements for each component of the derivative using the parameter shift rule (3) and returns the estimated gradient vector $\boldsymbol{g}$ as well as the vector $\boldsymbol{S}$ with the variances of the individual estimates of the partial derivatives.

Input: Learning rate α, starting point $\boldsymbol{\theta}_0$, min number of shots per estimation $s_{\min}$, number of shots that can be used in total N, Lipschitz constant L, running average constant μ, bias for gradient norm b

initialize: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$, $s_{\text{tot}} \leftarrow 0$, $s \leftarrow s_{\min}$, $\boldsymbol{\chi} \leftarrow (0, ..., 0)^T$, $\xi \leftarrow 0$, $k \leftarrow 0$
while $s_{\text{tot}} < N$ **do**
    $\boldsymbol{g}, \boldsymbol{S} \leftarrow Evaluate(\boldsymbol{\theta}, s)$
    $s_{\text{tot}} \leftarrow s_{\text{tot}} + 2s$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}$
    $\xi \leftarrow \mu\xi + (1-\mu)\|\boldsymbol{S}\|_1$
    $\boldsymbol{\chi} \leftarrow \mu\boldsymbol{\chi} + (1-\mu)\boldsymbol{g}$
    $s \leftarrow \left\lceil \frac{2L\alpha}{2-L\alpha} \frac{\xi}{\|\chi\|^2 + b\mu^k} \right\rceil$
    $\boldsymbol{s} \leftarrow \max(s, s_{\min})$
    $k \leftarrow k + 1$
end while

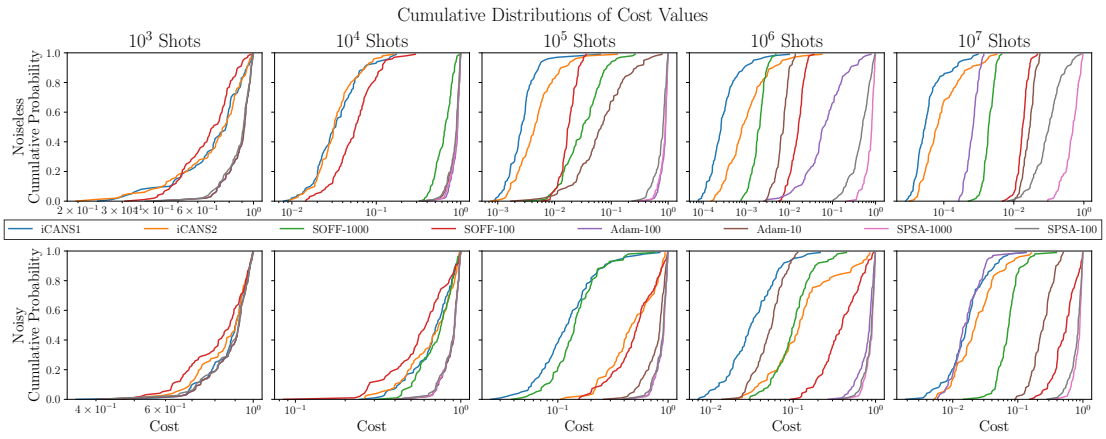

Figure 5: Comparison of performance for the varaitional compiling task across one hundred random starts. Each panel shows the cumulative probability distribution of the cost values acheived by each optimizer for a different value of total shots N. Note that the further to the left a curve is, the better the optimizer has minimized the cost.

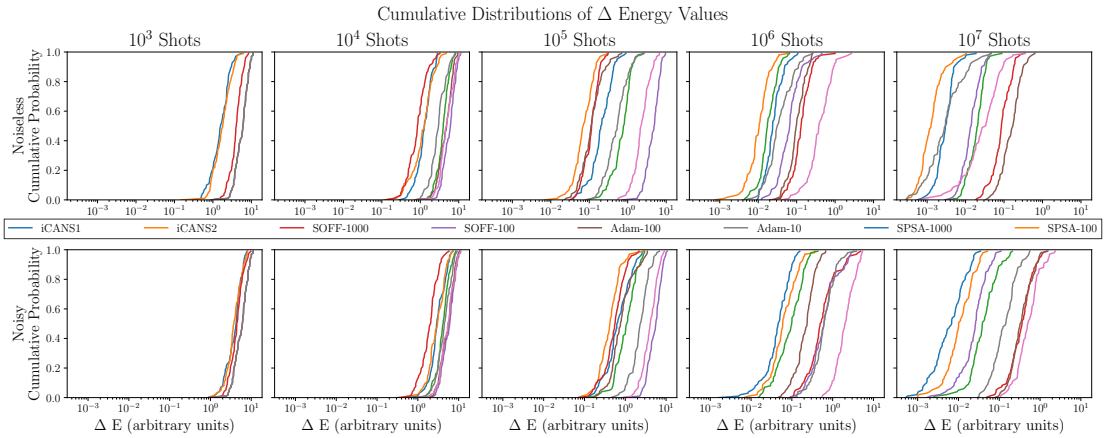

Figure 6: Comparison of performance for the Heisenberg spin chain VQE task across one hundred random starts. Each panel shows the cumulative probability distribution of the energy difference from the ground state acheived by each optimizer for a different value of total shots N. Note that the further to the left a curve is, the better the optimizer has minimized the energy.