Supplementary Information for "Efficient Online Quantum Circuit Learning with No Upfront Training"

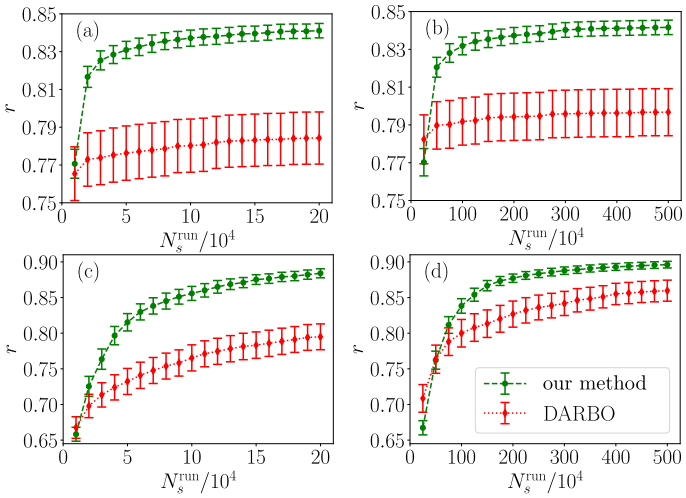
Tom O'Leary,[1,2] Piotr Czarnik,[3,4] Elijah Pelofske,[5] Andrew T. Sornborger,[6,7] Michael McKerns,[6,8] and Lukasz Cincio[1,7]

[1] *Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, USA.*
[2] *Department of Physics, Clarendon Laboratory, University of Oxford, Oxford, UK.*
[3] *Institute of Theoretical Physics, Jagiellonian University, Kraków, Poland.*
[4] *Mark Kac Center for Complex Systems Research, Jagiellonian University, Kraków, Poland*
[5] *Information Systems & Modeling, Los Alamos National Laboratory, Los Alamos, NM, USA.*
[6] *Information Sciences, Los Alamos National Laboratory, Los Alamos, NM, USA.*
[7] *Quantum Science Center, Oak Ridge, TN 37931, USA.*
[8] *The Uncertainty Quantification Foundation, Wilmington, DE 19801, USA.*

Supplementary Note 1: Additional Numerical Results

Here we gather additional numerical data to complement the results in the main text.

A. QAOA for 3-regular Max-Cut Weighted Graphs



Supplementary Figure 1. **A comparison of DARBO and the surrogate-based optimization for 3-regular 16-qubit Max-Cut problems using infinite-shot cost function estimates.** We compare the performance of optimization runs from the main text Fig. 2 using approximation ratios r, Eq. (1), computed with infinite-shot cost function values, instead of finite-shot estimates. In both cases, the approximation ratios of $C(\boldsymbol{\theta})$ are computed for the same best angles $\boldsymbol{\theta}$, which are found by optimization runs minimizing finite-shot $C(\boldsymbol{\theta})$ estimates. Panels (a,b) show results for $p = 2$ QAOA rounds and (c,d) results for $p = 10$. The r values for $N_s = 200$ optimization are gathered in panels (a,c) and $N_s = 5000$ in (b,d). The averaging and the error bar computation are performed as for Fig. 2 in the main text.

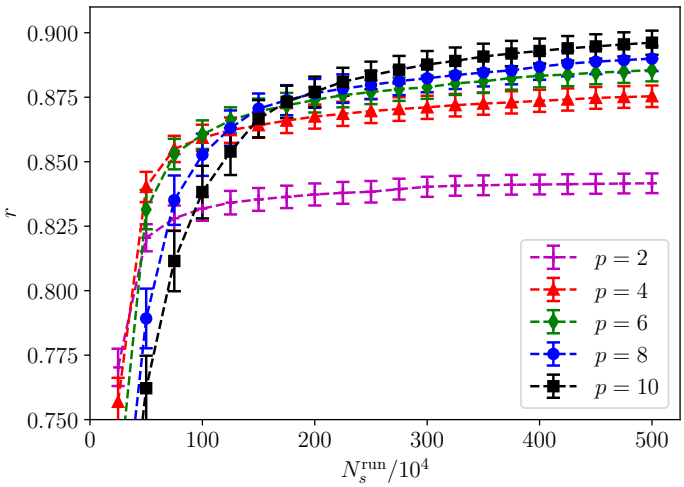
Here, to clarify the effects of finite N_s on the QAOA cost function estimates used by the optimization and to more robustly quantify the quality of the optimized angles, we plot the approximation ratios

$$r = \frac{C_{\max} - C(\boldsymbol{\theta})}{C_{\max} - C_{\min}}, \tag{1}$$

for the best angles found within the first N_s^{tot} shots of an optimization run, $\operatorname{argmin}\{C(\boldsymbol{\theta}) \,|\, \boldsymbol{\theta} \in \Theta\}$, using infinite-shot cost function $C(\boldsymbol{\theta})$ values. That is, the best angles, $\operatorname{argmin}\{C(\boldsymbol{\theta}) \,|\, \boldsymbol{\theta} \in \Theta\}$, are found using finite-shot $C(\boldsymbol{\theta})$ estimates

obtained during the optimization, while r is computed with the infinite-shot $C(\boldsymbol{\theta})$ for these angles. $C_{\max}$ and $C_{\min}$ are the maximum and minimum cost function values respectively.

In Supplementary Figure 1 we compare the our approach to DARBO, similar to Fig. 2 in the main text. Moreover, in Supplementary Figure 2 we analyze the improvement in results with increasing QAOA rounds, analogically to the main text Fig. 3. In both cases, we find that the results are qualitatively very similar to those in the main text obtained with finite-shot estimates.



Supplementary Figure 2. **The p convergence of the surrogate-based optimization for 16-qubit 3-regular Max-Cut problems with infinite-shot cost functions estimates.** We plot approximation ratio r, Eq. (1), for $N_s = 5000$ optimization runs from the main text Fig. 3. We use the infinite-shot cost function estimates that are computed for for the best angles found within the first N_s^{run} shots. The best angles are found using the finite-shot $C(\boldsymbol{\theta})$ estimates, as in Fig. 3 in the main text. The averaging, and the error bar computation are also performed as in Fig. 3 in the main text.

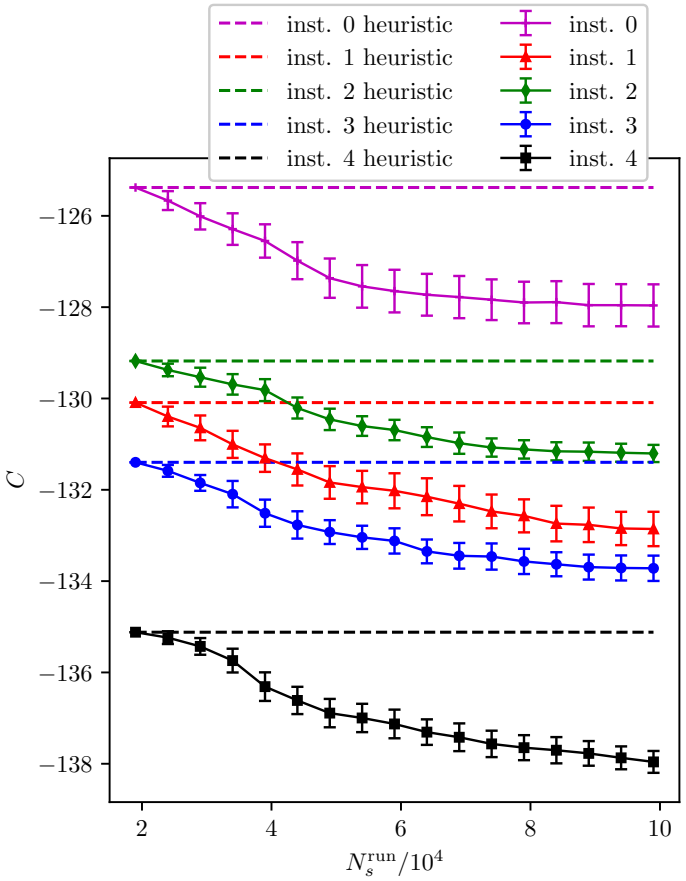
B. QAOA for Random Ising Model on Heavy-Hex Graphs

As with Supplementary Note 1 A, we use infinite-shot estimates of the cost function to more robustly quantify the improvement of the optimized angles with respect to the heuristic ones. In Supplementary Figure 3, we plot infinite-shot cost function estimates for the best angles found during the first N_s^{run} shots of the optimization runs from the main text Fig. 4, and for the heuristic angles $\theta_{\text{heur}}^{p=3}$. The best optimized angles are found according to the finite-shot, optimized estimates of the cost, similarly to Supplementary Note 1 A. We find that the infinite-shot cost function of the optimization runs improves upon the heuristic angles similarly to the results of Fig. 4 in the main text, which are obtained with finite-shot estimates.

Additionally, in Supplementary Table I, we compare the optimized costs C_{opt} in Eq. (12) in the main text for two different values of the matrix product state (MPS) simulation refinement parameter, $D = 512$, and $D = 1024$, which were used to perform the optimization. The main text results were obtained for $D = 1024$. In both cases, the cost function values averaged over the optimization runs agree within 95% confidence intervals estimated with the mean standard deviation. This indicates that the results are likely converged in D. We reach the same conclusion for MPS estimates of $C(\boldsymbol{\theta}_{\text{heur}}^{p=3})$.

Supplementary Note 2: Bond Dimension Convergence of the MPS Cost Function for the Hardware Runs.

In Supplementary Figure 4, we plot the MPS cost function for 9 representative `ibm_torino` optimization runs from the main text Fig. 5, and for the MPS refinement parameter $D = 256, 512, 1024, 2048$. We see that the results converge quickly with an increasing D. The observed convergence indicates that $D = 2048$, which is used in Fig. 5 in the main text, is large enough to provide reliable results.



Supplementary Figure 3. **A convergence of infinite-shot cost function estimates for matrix product state optimization of QAOA for heavy-hex 127-qubit random Ising models.**. Here we plot infinite-shot $C(\boldsymbol{\theta})$ estimates for the optimization runs and the heuristic angles, $\boldsymbol{\theta}^{p=3}_{\mathrm{heur}}$, from the main text Fig. 4. The solid lines are the cost estimates for the best angles found during the first N_S^{tot} shots of an optimization run and the dashed angles are the costs for the heuristic angles. The best optimized angles are found according to the finite-shot, optimized cost estimates, the same as in Fig. 4 in the main text. For each Hamiltonian instance we average over 32 runs and compute the error bars the same as in the main text Fig. 4.

Supplementary Note 3: Implementation of Surrogate-based Optimization using *mystic*

We use the *mystic* Python software to "wrap" an interface for surrogate-based learning [1, 2] around a `cost` function, where we first create a file-based archive to store the training `data` associated with calling the cost function.

```
# create a data archive, and associate it
# with a cost function
from mystic.cache.archive import file_archive, read
archive = read("truth.db", type=file_archive)
truth = WrapModel("truth", cost, nx=2*p, ny=None,
                  cached=archive)
```

This creates a callable object, `truth`, where `cached=archive` attaches a file-based `archive` named `truth.db` configured to log the inputs and output for every call of `truth`. The keywords `nx` and `ny` correspond to the dimensionality of the inputs and output, with `None` indicating a scalar.

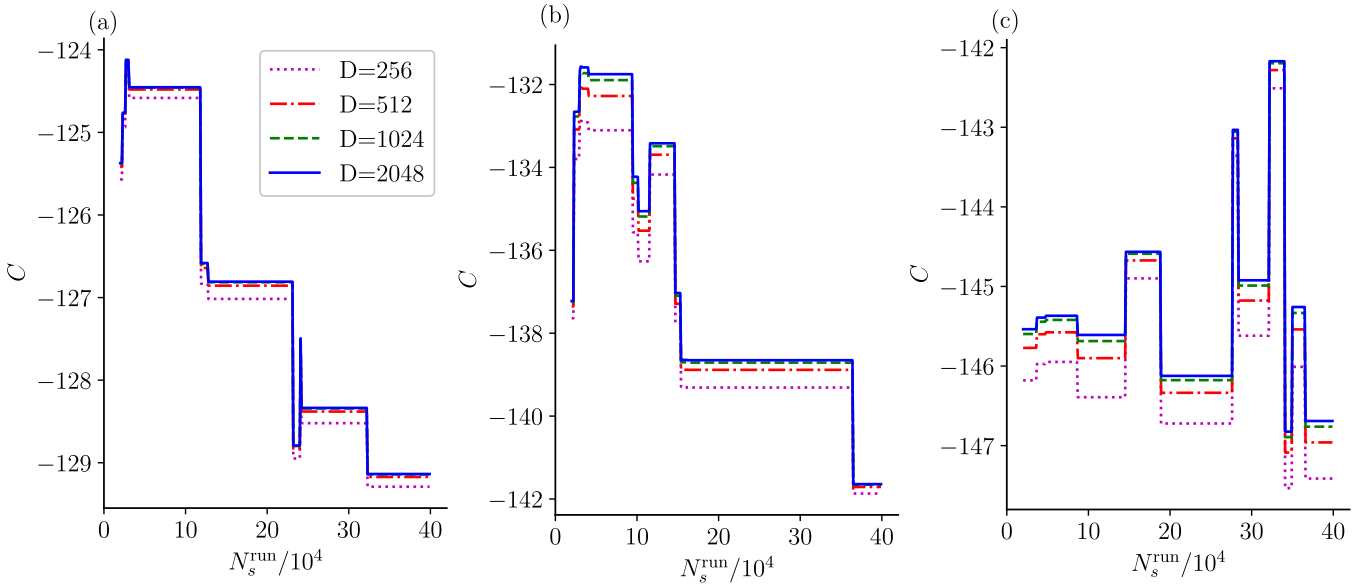
We then sample N_{init} initial datapoints, using uniform random sampling of the cost function, and build a classical `surrogate` from the training data using RBF interpolation.

```
# generate a dataset by sampling truth
data = truth.sample(bounds, pts=N_init)

# create an inexpensive surrogate for truth
```

instance	D	$C(\boldsymbol{\theta}^{p=3}_{\text{heur}})$	$C(\boldsymbol{\theta}_{\text{opt}})$
0	512	−125.4(1)	−129.1(3)
0	1024	−125.5(2)	−128.8(5)
1	512	−130.0(1)	−133.8(3)
1	1024	−130.0(2)	−133.7(4)
2	512	−129.3(2)	−131.9(2)
2	1024	−129.4(2)	−132.1(2)
3	512	−131.3(2)	−134.7(2)
3	1024	−131.4(1)	−134.6(3)
4	512	−135.0(1)	−138.6(3)
4	1024	−135.1(2)	−138.8(3)

Supplementary Table I. **A comparison of bond dimension $D = 512$ and $D = 1024$ matrix product state surrogate-based optimization for $p = 3$ QAOA and 127-qubit random heavy-hex Ising model.** Here we compare the final optimized cost function $C(\boldsymbol{\theta}_{\text{opt}})$ averaged over 32 $D = 512$ and $D = 1024$ optimization runs for 5 instances of the Hamiltonian. As a reference, we give $D = 512$ and $D = 1024$ $C(\boldsymbol{\theta}^{p=3}_{\text{heur}})$ estimates. The results for $D = 1024$ runs are plotted in the main text Fig. 4, while the $D = 512$ results were obtained in the same way, with the only difference being D value. For a reference, we also list here $D = 512$ and $D = 1024$ estimates of $C(\boldsymbol{\theta}^{p=3}_{\text{heur}})$. Here, $C(\boldsymbol{\theta}^{p=3}_{\text{heur}})$ is evaluated once per an optimization run, as $\boldsymbol{\theta}^{3}_{\text{heur}}$ is included among runs' initial angles, and is averaged over all its evaluations. We compute the error bars as the mean standard deviation multiplied by a factor of 2. All the cost functions are evaluated with $N_s = 1000$ shots.



Supplementary Figure 4. **Convergence of the matrix product state (MPS) cost function with MPS bond dimension, D, for representative `ibm_torino` hardware optimization runs.** In this figure, we plot the MPS infinite-shot cost function C for hardware runs nb. 0 from the main text Fig. 5, and $p = 3$ (a), $p = 4$ (b), $p = 5$ (c). The $D = 2048$ results are plotted in insets of Fig. 5 in the main text.

```
kwds = dict(smooth=0.0, noise=0.0, method="thin_plate")
surrogate = InterpModel("surrogate", nx=2*p, ny=None,
                        data=truth, **kwds)
```

Here, `InterpModel` creates a callable object that performs RBF interpolation that uses `mystic.math.interpolate.interpf` [1] on the data associated with `truth`. A dictionary of hyperparameters, `kwds`, is used to configure the interpolation, and thus the surrogate. We choose a thin-plate basis function, where the interpolation is constrained to go through the data (i.e. no smoothing, and no noise). These parameters are not chosen through pre-training, but instead because they are the most robust settings that enable the surrogate to reproduce the training data exactly. The keywords `nx` and `ny` are identical to those in `WrapModel`, while `data=truth` associates the surrogate with the archive of training data in `truth.db`.

The archive of training `data` is of the form $\{(C(\boldsymbol{\theta}), \boldsymbol{\theta}) \,|\, \boldsymbol{\theta} \in \Theta\}$, with Θ being a set of all parameters for which the device cost function C was evaluated, and a surrogate C^{surr} that has not yet been fit to the training data. The remainder of our approach iteratively fits the surrogate to the training data. Within each iteration, we perform an optimization that solves for the minimum $\boldsymbol{\theta}_{\text{cand}}$ of the updated C^{surr}, and compares $C^{\text{surr}}(\boldsymbol{\theta}_{\text{cand}})$ to $C(\boldsymbol{\theta}_{\text{cand}})$.

Specifically:

1. Build an initial set of training data from the quantum computer in the form of $\{(C(\boldsymbol{\theta}), \boldsymbol{\theta})\}$.
2. Create an `InterpModel` to fit to the training data.
3. Continue to iterate the following, until termination is reached:
   (a) Fit the surrogate to the training data.
   (b) Find the surrogate minimum using Differential Evolution.
   (c) Evaluate the cost function at the surrogate minimum.
   (d) Update the training data with the new evaluation of cost.

We perform the minimization on the surrogate with the Differential Evolution algorithm `diffev2` from *mystic* [1, 2].

```
from mystic.abstract_solver import AbstractSolver
from mystic.termination import EvaluationLimits
loop = AbstractSolver(3) # nx
loop.SetTermination(EvaluationLimits(maxiter=500))
while not loop.Terminated():

    # fit the surrogate to data in truth database
    surrogate.fit(data=data)

    # find the minimum of the surrogate
    results = diffev2(lambda x: surrogate(x), bounds,
                      bounds=bounds,npop=20*p, gtol=500,
                      ftol=5e-4, full_output=True)

    # evaluate truth at the same input
    # as the surrogate minimum
    xnew = results[0].tolist()
    ynew = truth(xnew)
```

We configure `diffev2` to have `bounds` as described in Section "Hardware Implementation" in the main text, a population `npop` of $20p$ for p QAOA layers, and a termination when the cost function does not change more than `ftol` over `gtol` iterations. We note that we rescale the cost function by the number of graph vertices n, to ensure approximately constant scaling of the allowed cost function values with n. We set this termination condition for our real device optimizations to not exceed N_{it} iterations (including the initial sampling of points). The Differential Evolution optimizer with the above hyperparameters is configured to greatly increase the likelihood that the surrogate global minimum is found at each attempt. We find that the cost of evaluating the surrogate is negligible in comparison to the evaluation of truth. This suggests that, in our setting, the choice of optimizer and optimizer hyperparameters are secondary considerations beyond defining a limit for the maximum number of evaluations the global optimizer can attempt.

Supplementary References

[1] Michael McKerns, Patrick Hung, and Michael Aivazis, "mystic: highly-constrained non-convex optimization and UQ," (2009-).
[2] M. McKerns, L. Strand, T. Sullivan, A. Fang, and M.A.G. Aivazis, "Building a framework for predictive science," Proceedings of the 10th Python in Science Conference (2011), 10.48550/arXiv.1202.1056.