Deep recurrent networks predicting the gap evolution in adiabatic quantum computing

Naeimeh Mohseni,[1, 2] Carlos Navarrete-Benlloch,[3, 4, 1] Tim Byrnes,[5, 6, 7, 8, 9, *] and Florian Marquardt[1, 2]

[1] *Max-Planck-Institut für die Physik des Lichts, Staudtstrasse 2, 91058 Erlangen, Germany*
[2] *Physics Department, University of Erlangen-Nuremberg, Staudtstr. 5, 91058 Erlangen, Germany*
[3] *Wilczek Quantum Center, School of Physics and Astronomy, Shanghai Jiao Tong University, Shanghai 200240, China*
[4] *Shanghai Research Center for Quantum Sciences, Shanghai 201315, China*
[5] *New York University Shanghai, 1555 Century Ave, Pudong, Shanghai 200122, China*
[6] *State Key Laboratory of Precision Spectroscopy, School of Physical and Material Sciences, East China Normal University, Shanghai 200062, China*
[7] *NYU-ECNU Institute of Physics at NYU Shanghai, 3663 Zhongshan Road North, Shanghai 200062, China*
[8] *National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan*
[9] *Department of Physics, New York University, New York, NY 10003, USA*

(Dated: September 20, 2021)

One of the main challenges in quantum physics is predicting efficiently the dynamics of observables in many-body problems out of equilibrium. A particular example occurs in adiabatic quantum computing, where finding the structure of the instantaneous gap of the Hamiltonian is crucial in order to optimize the speed of the computation. Inspired by this challenge, in this work we explore the potential of deep learning for discovering a mapping from the parameters that fully identify a problem Hamiltonian to the full evolution of the gap during an adiabatic sweep applying different network architectures. Through this example, we find that a limiting factor for the learnability of the dynamics is the size of the input, that is, how the number of parameters needed to identify the Hamiltonian scales with the system size. We demonstrate that a long short-term memory network succeeds in predicting the gap when the parameter space scales linearly with system size. Remarkably, we show that once this architecture is combined with a convolutional neural network to deal with the spatial structure of the model, the gap evolution can even be predicted for system sizes larger than the ones seen by the neural network during training. This provides a significant speedup in comparison with the existing exact and approximate algorithms in calculating the gap.

I. INTRODUCTION

Machine learning based on neural networks has demonstrated significant predictive capability for many challenging problems [1, 2]. In particular, its application in studying quantum many-body systems has attracted significant attention in the last few years [3]. A few notable successes include the use of neural networks in identifying quantum phases of matter and learning phase transitions [4–8], quantum state tomography [9, 10], enhancing quantum Monte Carlo methods [11, 12], solving optimization problems [13, 14], quantum control [15], the efficient representation of quantum states [16, 17], and tackling quantum many-body dynamics [17–20].

Adiabatic quantum computing (AQC) is an example of a quantum many-body dynamical process, and was proposed as an approach for solving optimization problems [21]. The sweep time in AQC for which adiabaticity can be achieved is proportional to a negative power of the minimum energy gap between the two lowest energy levels during the sweep [22–24]. Therefore, one way of optimizing the computation time is by spending the majority of the evolution time in the vicinity of the anti-crossing. Creating such an annealing schedule requires prior knowledge of the gap structure, which is known to be as hard as solving the original problem [25]. There are a handful of cases for which analytical expressions for the spectral gap exist [22, 26–28], but mostly the gap can only be determined numerically either based on exact diagonalization or approximate algorithms [29, 30] which are limited to relatively small system sizes. Therefore, in general there is no intuition about how the spectral gap is related to

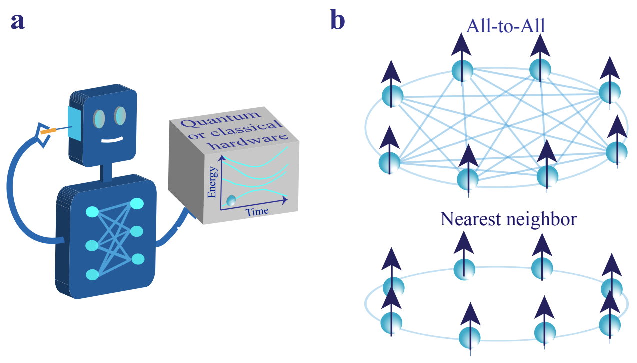

FIG. 1. (a) A neural network learns the gap dynamics by observing the gap dynamics generated by a classical computer or a quantum computer. For the purposes of this work, we train the network on the data generated by a classical computer. (b) Spin models studied in this work that are taken to be the problem Hamiltonian in AQC. Either all spins or only nearest neighbors are connected with random couplings.

* Corresponding author: tim.byrnes@nyu.edu

the structure of the problem Hamiltonian.

It is of interest to characterize the complexity of learning the full gap evolution during the adiabatic sweep by applying a classical machine (neural network) that is trained on the data generated by a quantum computer or numerical simulations (Fig. 1 (a)). In the former case we have a hybrid quantum-classical algorithm. Such algorithms have recently attracted a lot of interest and have been examined in different contexts [18, 31, 32], as they can be applied for regimes that the run-time of the exact numerical simulations are prohibitive but the network can be still trained on the data generated via experiment. One should note that in AQC a highly precise estimation of the gap is not necessary as long as it is not overestimated significantly, since diabatic excitations are not produced as long as the adiabatic sweep time is longer than that required by the gap. It is also interesting to investigate how the difficulty of the task is related to the complexity of the problem Hamiltonian and what are the cases for which a neural network can assist in gap approximation.

Inspired by these open questions, here we explore the power of a neural network in discovering a mapping from the parameters that fully specify the adiabatic Hamiltonian to the evolution of the gap by having it observe the dynamics of the gap for many different realizations of a random problem Hamiltonian. For the purposes of this work, we train the neural network on the data generated from numerical simulations rather than quantum hardware. However, our methods could equally be applied in a quantum-classical hybrid scenario. As for the problem Hamiltonian in AQC, we consider spin models in two extreme limits: either all spins are connected, or only nearest neighbors in 1D are (Fig. 1 (b)). We then compare the standard fully-connected neural networks (FCNNs) against a particular type of recurrent neural networks called long short-term memory (LSTM) for the task of learning the gap evolution during the sweep for both of these spin models. We observe that our neural networks are able to learn the mapping for the nearest neighbor connected model with a good precision, but they fail for the all-to-all connected case. We also find that LSTM networks, which are typically used for sequence processing and prediction, including audio signal analysis [33] and language translation [34], excel at this task. These architectures are known to be good at capturing both long-term and short-term dependencies in sequences. This characteristic is extremely useful as it gives the LSTM network the power to handle complex dynamics.

Remarkably, we demonstrate for the nearest neighbor connected model that a convolutional long short-memory (CONVLSTM) [35] can predict the gap for system sizes larger than those that the network has been trained on. This architecture combines a convolutional neural network (CNN) to deal with the spatial structure of the input with the LSTM that tracks the time evolution.

While throughout this work we concentrate on the particular task of gap prediction in AQC, our study provides insight for more general many-body dynamics. We conclude that an important limiting factor for the learnability of such dynamics is the way in which the number of parameters needed to identify the Hamiltonian scales with the system size. Moreover, our study demonstrates the promise of convolutional recurrent neural networks for extrapolating the dynamics of inhomogeneous time-dependent quantum many-body models beyond the system sizes that they are trained on.

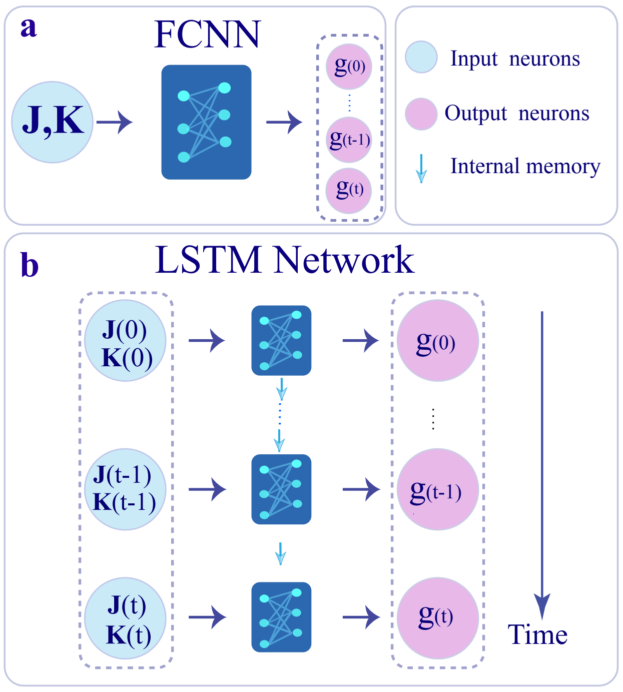

FIG. 2. Schematic representation of the neural networks considered in this work. (a) Fully connected neural network (FCNN) with the bare parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ as the input and the full gap evolution as the output. (b) The long short-memory (LSTM) network with $\boldsymbol{J}(t) = \lambda(t)\boldsymbol{J}$ and $\boldsymbol{K}(t) = \lambda(t)\boldsymbol{K}$ as the input and the gap as the output denoted by $g(t)$. Arrows between the modules (dark blue cells) indicate the content of the internal neural memory being passed to the next time step.

II. PROBLEM DEFINITION

In this section, we define the models that we explore in this work. We also introduce the neural network architectures that we apply and explain how we train them.

A. Model

We consider the AQC Hamiltonian defined as

$$H = [1 - \lambda(t)]H_0 + \lambda(t)H_{\mathrm{p}}, \tag{1}$$

with

$$H_{\mathrm{p}} = \sum_{i,j=1}^{M} J_{ij}\sigma_i^z\sigma_j^z + \sum_{i=1}^{M} K_i\sigma_i^z, \tag{2a}$$

$$H_0 = -\sum_{i=1}^{M} \sigma_i^x, \tag{2b}$$

where σ_i^α with $\alpha = x, z$ are Pauli operators, and J_{ij} and K_i are random coefficients that identify the problem Hamiltonian. $\lambda(t)$ is a time-varying parameter that is swept from 0 to 1. The system is initially prepared in the ground state of H_0 and the Hamiltonian gradually transitions to the desired problem Hamiltonian H_{p}. Based on the adiabatic theorem if one performs the sweep sufficiently slowly, the system will remain in its instantaneous ground state throughout the evolution [21]. In this work we consider the adiabatic schedule $\lambda(t) = t/\tau$ where τ denotes the adiabatic duration.

As mentioned previously, our goal is to find a mapping from the parameters (J_{ij}, K_i), which we collect in a matrix $\boldsymbol{J}$ and a vector $\boldsymbol{K}$, to the full gap evolution. In addition to assisting in the search for the best annealing schedule, such a mapping may also help to classify harder problem instances from easier ones in terms of the gap. We study two limiting cases of problem Hamiltonians: i) All-to-all connected spin models and ii) 1D nearest neighbor connected models.

B. Neural network architectures

We apply three neural network architectures for our goal: an FCNN, an LSTM network, and a CONVLSTM network. For the FCNN architecture, we feed into the neural network the parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ that fully identify the problem Hamiltonian and the neural network provides as output the full gap evolution during the sweep (Fig. 2 (a)), which we signify by

$$(\boldsymbol{J}, \boldsymbol{K}) \xrightarrow{\text{FCNN}} g(t), \tag{3}$$

where $g(t)$ denotes the full gap evolution during the adiabatic sweep.

In the LSTM architecture, the input is not just the bare parameters that identify the problem Hamiltonian, but the effective contribution of these parameters during the whole sweep, namely $\lambda(t)\boldsymbol{J}$ and $\lambda(t)\boldsymbol{K}$ (Fig. 2 (b)). These coefficients effectively identify the full adiabatic Hamiltonian during the sweep. In this case, then we have

$$(\lambda(t)\boldsymbol{J}, \lambda(t)\boldsymbol{K}) \xrightarrow{\text{LSTM}} g(t). \tag{4}$$

The most important difference between these two architectures is that the LSTM receives as input the effective contribution of the parameters at each time step and computes the gap for that time step, eventually working its way sequentially through the whole time interval. In contrast, the FCNN has to work out the full gap evolution at once. In addition, as shown in Fig. 2(b), the LSTM architecture is composed of modules (dark blue cells). Each module is made of a few gates (see Sec. II of the Supplemental Material for more details) which decide on the flow of information in and out of each module at each time.

To study the possibility of extrapolating the predictions to system sizes beyond what the neural network has been trained on we also apply a CONVLSTM network [35]. This architecture is designed for data with spatio-temporal input [36]. It combines a CNN to deal with the spatial structure of the input with the LSTM that tracks the time evolution. CNNs can be applied for variable input size therefore helping to scale up the predictions to larger sizes [1, 37] (See Supplemental Material Secs. I and II for more details). Inspired by this feature and using an appropriate preparation of the input such that we can present properly the spatio-temporal structure of the input to the network we explore the power of the CONVLSTM in predicting the gap for larger system sizes than that it has been trained on. The input and the output of the network in this case are signified as

$$(\lambda(t)\boldsymbol{J}(x), \lambda(t)\boldsymbol{K}(x)) \xrightarrow{\text{CONVLSTM}} g(t), \tag{5}$$

where $\boldsymbol{J}(x)$ and $\boldsymbol{K}(x)$ should properly present the spatial structure of the problem Hamiltonian as we explain in more detail in Sec. IV.

C. Training

To train the neural network, we generate a set of random parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ for the particular system size of interest. We then diagonalize the Hamiltonian during the sweep to calculate the first two eigenvalues and therefore the full gap evolution. All these parameters are taken from a uniform distribution in the interval $[-1, 1]$. To improve the performance of the neural network at regions where the gap is smaller, we train it on the $\log(g(t) + 1)$ rather than $g(t)$.

Out of the generated random instances, we keep a set to evaluate the neural network, which we call the test set, and a set for validation. The validation set is used to fine tune the hyperparameters of the neural network, but no training occurs on this set. The remaining instances are used for training. To evaluate the performance of the neural network, we calculate the mean square error MSE on our test set defined as

$$\mathrm{MSE} = \langle |g_{\mathrm{true}}(t) - g_{\mathrm{predict}}(t)|^2 \rangle_{n,t}, \tag{6}$$

where the g_{true} and the g_{predict} denote the true gap calculated by diagonalizing the Hamiltonian and the predicted gap obtained by the neural network, respectively. The average is taken over the number of instances n and time.

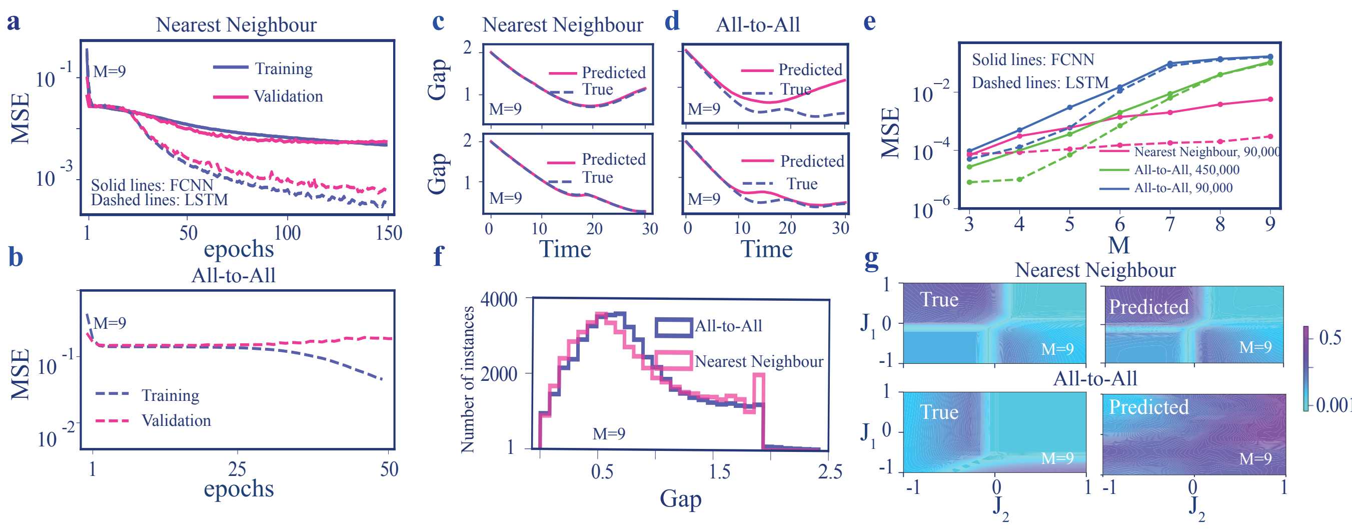

FIG. 3. Comparing the performance of the LSTM network and FCNN in predicting the gap evolution during the adiabatic sweep for the all-to-all and the nearest neighbor connected models. Validation MSE versus epochs for system size $M = 9$ applying the FCNN (solid lines) and LSTM network (dashed lines) for (a) the nearest neighbor and (b) all-to-all connected model. 90,000 samples are used to train the neural network and 10,000 samples are used for the validation. Predicted gap (solid line) and exact gap (dashed line) in terms of time for a few typical problem instances for the (c) nearest neighbor connected model and (d) all-to-all connected models. (e) The error MSE versus system size on the test set (containing 1000 samples for each system size) for the nearest neighbor and all-to-all connected models applying both FCNN (solid curves) and LSTM network (dashed curves). The numbers on the legend denote the training set size. The shown points are average over 5 attempts for training. (f) Histogram of the gap during the whole sweep for both the nearest neighbor and all-to-all connected models. (g) The true and predicted minimum gap versus couplings J_1 and J_2, where all other couplings and local terms are fixed.

III. GAP EVOLUTION LEARNABILITY

In this section, we study the learnability of the gap applying the different neural network architectures that we discussed above. We study the performance of our neural network architectures in predicting the gap for spin models in two extreme limits: either all spins are connected or only nearest neighbors in 1D. We inspect how the prediction accuracy scales with system size for both models applying our different network architectures.

In Fig. 3(a) and (b), we show the MSE over training and validation sets in terms of epochs for the nearest neighbor and all-to-all connected models. The number of epochs indicates the number of times that the network has seen all the training instances. For the nearest neighbor connected model, it can be seen that for both architectures the error decreases with the number of epochs. However, it is evident that the LSTM network performs better (Fig. 3(a)). In contrast, for the all-to-all connected model, the network overfits (Fig. 3(b)). For this model we just showed the results using LSTM network as we observed the same behavior applying the FCNN as well. Investigating different factors such as the network size and the training set size, we learned that the latter is the main bottleneck in this case.

In Figs. 3(c) and (d), we show the predicted and the true gap for a few typical instances applying LSTM network. It is clear that our network fails to predict the gap for the all-to-all connected instances while successfully predicts the gap with a high accuracy for the nearest neighbor connected instances.

In Fig. 3 (e), we study how the error scales with the system size for a fixed number of training samples specified in the legend. The shown errors are found by averaging over 1000 test instances. Let us focus first on the nearest neighbor connected model. It is obvious that the LSTM network has a considerably higher precision and the error in prediction scales more favorably in terms of system size for this architecture in comparison with the FCNN (compare the pink solid line with the pink dashed line). We attribute this to the fact that the LSTM architecture has memory and is able to record both long and short-term dependencies. This architecture has the built-in the notion of causality, while the FCNN needs to learn causality on its own as it has no notion of time. Applying the same number of training samples for the all-to-all connected model (blue lines), the error grows more dramatically with system size in comparison with the nearest neighbor connected model. The error can be decreased by increasing the training set size, but even a factor 5 (green lines) is not enough to achieve reasonable accuracy for larger system sizes (say $M > 6$). This implies that the number of samples required to train the network explodes with the system size when aiming at a given error in the predictions. As a consequence of this explosion, the LSTM network does not seem to show a better performance in comparison with the FCNN for

$M > 5$ in the figure when 90,000 instances are used for training (compare dashed and solid blue lines).

We have explored some reasons why the network fails for the all-to-all connected model, while it succeeds for the nearest neighbor model when for a given system size M the Hilbert space dimension is the same for both models. We conjecture the main reason is due to the way in which the parameter space size scales with the system size for each model: linearly for the nearest neighbor model and quadratically for the all-to-all connected one. Note that by parameter space size we mean the number of parameters that identify the model. One may think that another reason might be that the nearest neighbor connected model can be also simpler in terms of the gap size and complexity of the gap trajectories. To investigate this, we compare the gap size during the sweep for both the nearest neighbor and the all-to-all connected models for system size $M = 9$ in Fig. 3 (f). It is apparent that, in general the typical size of the gap is similar for both models. Moreover, in Fig. 3 (g), for a typical problem instance we compare the dependence of the minimum gap with two couplings J_1 and J_2, with all other parameters fixed. As can be seen, the pattern of the minimum gap in the all-to-all connected model does not appear to be more complicated than the nearest neighbor connected model. However, network fails in predicting the minimum gap for the all-to-all connected model. While Fig. 3 (g) is for a single problem instance, we observed the similar complexity for the minimum gap for both models via inspection of ~ 20 randomly generated problem instances. Another potential reason is that the local nature of the connectivities in the nearest neighbor model can make it easier for the network to learn the dynamics of the gap. If that were to be true, then one should expect an improvement by encoding the all-to-all connected model into a local model. In Sec. V, we introduce such an encoding through the so called Lechner-Hauke-Zoller (LHZ) mapping [38], and find the same scaling of the error with the system size.

In order to provide further evidence for our claim that the main bottleneck for a successful training comes from how the parameter space scales with the system size, here we compare the performance of our neural network for both spin models with sizes for which they contain the same number of parameters. In particular, the number of couplings for the $M = 5$ all-to-all connected model is 10, the same as for the $M = 10$ nearest neighbor connected model. Using the same number of training samples for both models, we observe that the network reaches MSE $= 0.0006$ for the all-to-all connected model and MSE $= 0.0002$ for the nearest neighbor connected model. These errors are comparable, and the performance for the nearest neighbor model is just slightly better. This result supports our conjecture.

As a last remark of this section, one may argue that it might be easier for the network to learn the times for which the gap becomes small, rather than the full gap evolution. We have also investigated this and observed that the network still fails for the all-to-all connected model. Our conjecture is that this is again a consequence of the quadratic scaling of the parameter space size with the system size.

IV. EXTRAPOLATION

In this section, we explore the possibility of the network to predict the gap for system sizes beyond those which it is trained. As indicated already, the potential architectures for this goal are CNNs, which can be applied for an input with variable size. These architectures are known to be good for extracting features on local models. Therefore, we apply to the nearest neighbor connected model the CONVLSTM architecture which is designed for sequence prediction problems with spatial structure. This network extends the fully connected LSTM architecture to have a convolutional structure in both the input-to-module (dark blue sheet Fig. 4(a)) and module-to-module (light blue sheet Fig. 4(a)) transitions.

In Fig. 4(a), we show the input of the network with a 1D spatial structure and containing three features at each site. The first feature represents the corresponding local parameter $\boldsymbol{K}$ at each site. The second and third ones represent the couplings $\boldsymbol{J}$, arranged so as to emphasize that each site is connected to its two nearest neighbors. See Supplemental Material Sec. III for more details on the technical implementation and the layout of the network.

We train the network on system sizes $M \in [3, 9]$, and then evaluate it on the test samples with system sizes $M \in [3, 22]$. We observed that our CONVLSTM network succeeds in predicting the gap both for the system sizes that it has been trained on and larger system sizes than that. In Fig. 4 (b), we show the predicted gap (solid line) and the true gap (dashed line) as a function of time for three typical problem instances with $M = 20, 21, 22$. As can be seen, the network is able to approximate the gap during the evolution with a good precision. We also find that there is a correlation between the size of the gap and the accuracy of the network. In Fig. 4 (c), we show the true gap against the predicted gap for the full gap evolution for 20 test instances. It is clear that close to the regions where the gap is smaller, the performance of the network is less accurate, even on an absolute level.

In Fig. 4 (d), we show how the prediction error in prediction scales with the system size. The highlighted region marks the system sizes that the network has not been trained on. We have not been able to conclude whether the error scales polynomially or exponentially, since both functions fitted relatively well.

In the previous section, we observed for the all-to-all connected model, using 90,000 samples for training, the network is able to learn the gap on small system sizes with $M < 7$. We are interested to study whether a CONVLSTM network that is trained on that small sizes can still extrapolate the predictions to the larger system sizes. To apply CONVLSTM network for the all-to-all connected model, we need to map it to a local model.

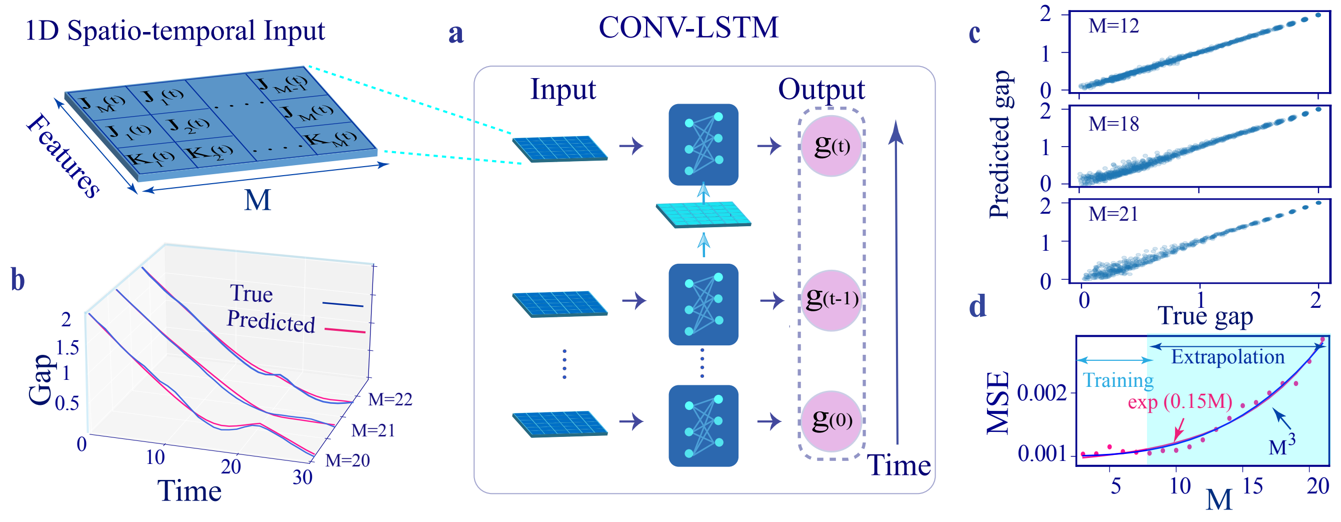

FIG. 4. Extrapolating the gap to the larger system sizes applying 1D-CONVLSTM network for the nearest neighbor connected model. (a) Schematic representation of the CONVLSTM with spatio-temporal inputs (light blue sheets). Arrows between the modules (dark blue cells) indicate the content of the internal neural memory being passed to the next time step and the dark blue sheet denotes a convolutional structure for module-to-module transitions. (b) Predicted (solid line) and true gap (dashed line) in terms of time for three typical problem instances. (c) The predicted gap versus the true gap for the full adiabatic evolution. For each system size the plot contains 20 problem instances. (d) The MSE error in predicting the gap versus system size. Network is trained on system sizes with $M \in [3, 9]$ and is evaluated on test samples with $M \in [3, 22]$. The shown errors are averaged over 200 random problem instances for each system size. The highlighted region marks the system sizes that the network has not been trained on. 90,000 samples are used to train the network.

This is because CNNs are good at extracting features on the local models as neurons of different layers are locally connected. In the next section we map the all-to-all connected model to a local model and explore the potential of CONVLSTM for extrapolating the predictions to larger system sizes.

V. MAPPING THE ALL-TO-ALL CONNECTED MODEL TO A LOCAL MODEL

In this section we first investigate whether the highly non-local nature of the all-to-all connected model is one of the reasons that makes it hard for the network to predict the gap. To understand this, we apply the LHZ mapping [38] to encode this model into a model with only local connectivities and check if the network performs better. LHZ maps the graph of the M all-to-all connected logical qubits (Fig. 5 (a)) onto a planar graph with $N_p = M(M-1)/2$ physical qubits and only local connectivities (Fig. 5 (b)). Within this mapping, the original problem Hamiltonian (2a) can be encoded into the following Hamiltonian in the physical qubit basis

$$H_{\mathrm{p}}^{\mathrm{LHZ}} = \sum_{k=1}^{N_p} J_k \sigma_k^z - C \sum_{\langle i,j,k,l\rangle} \sigma_i^z \sigma_j^z \sigma_k^z \sigma_l^z, \qquad (7)$$

where $\langle i, j, k, l\rangle$ denotes sum over nearest neighbor spins. Energy penalties in the second term which involve $M - N_p + 1$ four-body interactions (the four qubits around each small pink circle shown in Fig. 5 (b)) are introduced to enforce the system in the low energy sub-space. In the bottom row only three qubits are connected, for which three-body constrains are introduced instead.

Single-body terms in the original model, which correspond to a local field acting on the logical qubits, can be also implemented adding an auxiliary logical qubit (light blue qubit in Fig. 5 (a)) fixed to state $|0\rangle$, which is an eigenstate of σ_z with eigenvalue $+1$. Interaction of this auxiliary logical qubit with the rest of the qubits implements these local terms, which correspond to M extra physical qubits (light blue qubits in Fig. 5 (b)) in the LHZ architecture [38].

In Fig. 5 (c), we compare the performance of the LHZ model with the originally all-to-all connected model applying the standard LSTM network. 90,000 instances are applied to train the network for each model and system sizes up to $M = 5$, for which the LHZ model already contains 15 physical qubits. We also show the results for the nearest neighbor connected model for the purpose of comparing the scaling. We observe that for the LHZ model the error still scales exponentially with the system size. While we have not been able to analyze for larger M values, these results seem to suggest that the locality of the connectivities should not improve the way that error scales with the system size.

In Fig. 5 (d), we compare the typical size of the gap for both models. As can be seen, the LHZ model

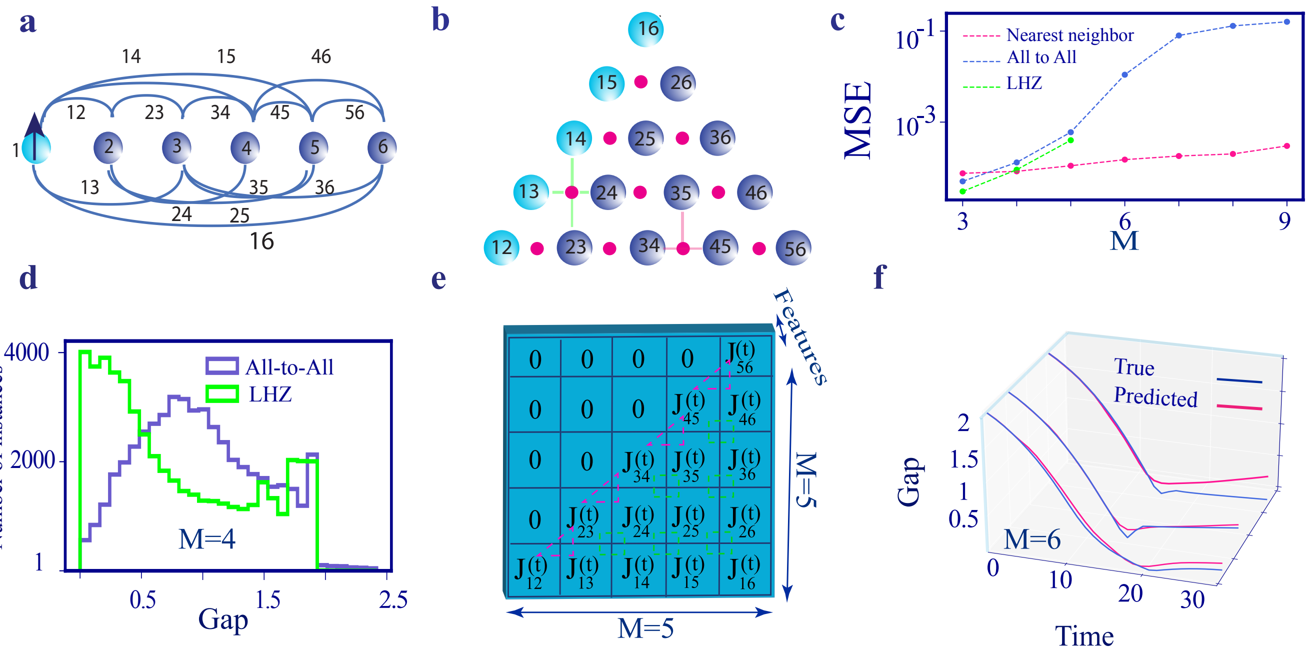

FIG. 5. The power of the LSTM network in predicting the full gap evolution during an adiabatic sweep for the LHZ model. (a) An all-to-all connected model with system size $M = 5$ where the light blue circle shows an auxiliary logical qubit (fixed to 1) to implement single-body terms. (b) The LHZ architecture where the light blue circles show physical qubits that encode single-body terms. The four qubits around each small pink circle (plaquette) consist four-body constraints. (c) The error for predicting the gap evolution averaged over 1000 realizations for all three investigated models versus system size applying LSTM network. 90,000 instances are used to train the network for all the shown models and system sizes. (d) Histogram of the gap during the sweep for the all-to-all connected and LHZ models for system size $M = 4$. (e) Spatio-temporal input of the 2D-CONVLSTM containing one feature at each site which presents the couplings. The couplings are arranged on the shown square such that they respect the connectivities in (b). The dashed squares and three-angles denote connections between qubits that identify the three-body and four-body constraints on LHZ model. The square at top left shows the second feature which identifies for the network where a qubit exists. (f) Exact gap and predicted gap (applying 2D-CONVLSTM) in terms of time for three particular problem instances with size $M = 6$ where network has not been trained on.

has substantially more instances with smaller gap size as compared to the all-to-all connected model, but still the network achieves a comparable or even better precision. This confirms our conjecture that the main bottleneck is not the size of the gap, but rather the way in which the parameter space size scales with the system size.

Now we explore whether mapping the all-to-all connected model to a local model and applying a CONVLSTM network helps to extrapolate the predictions to larger system sizes. For this case we need a 2D-CONVLSTM as the LHZ model has a 2D spatial structure. In Fig. 5 (e), we show the spatio-temporal input of our 2D-CONVLSTM for the case with $M = 5$. Each site contains one feature which represents the coefficients $\boldsymbol{J}$. These couplings are arranged on the shown square such that they respect the connectivities in the LHZ model. Dashed squares and triangles mark qubits around each small pink circle in Fig. 5 (b). We train the network on system sizes $M \in \{3, 4, 5\}$ and evaluate it on system size $M = 6$ (Fig. 5 (e)), which is the largest size (includes 21 physical qubits) that we are able to generate a few samples for. As can be seen, the network is still able to predict the gap for a larger system size that it has not been trained on, but not with a high accuracy such as in the nearest neighbor connected model. Due to the numerical limitations we have not been able to evaluate our network for larger system sizes, but considering the low precision for $M = 6$, we expect the network to fail for the larger sizes. We attribute this partially again to the fact that the quadratic scaling of the parameter space with system size necessitates exponentially growing resources for training. However we expect our 2D CONVLSTM succeeds in predicting the dynamics of 2D models for which the parameter space size scales more favorably with the system size.

VI. SPEEDUP

In this section we discuss the potential speedup that can be achieved by employing the network for the nearest-neighbor model, for which we have shown the network succeeds in making predictions and in extrapolating.

Supervised training of neural networks can be motivated in many ways, e.g. an explicit algorithm to turn the input into the desired output may not even be known in principle (as in image classification). However, for a scenario like the one discussed here, a numerically exact algorithm exists. One can then think of at least three reasons why training and deploying a network might still be beneficial: (i) the run-time of the network is so much shorter than the run-time of the exact algorithm, and we will use it for prediction on so many problem instances, that it is worth to invest the numerical effort needed to generate the large amount of training examples in the first place; (ii) the run-time of the algorithm scales very unfavourably with system size and the network is able to make reasonably accurate predictions also for larger sizes, even if it was not trained on those; (iii) there is a regime where the run-time of the exact algorithm is prohibitive, but data may be generated in another way (e.g. via experiments), and the network can still be made to learn in this regime and be used for prediction. We will discuss the last point (training from experiments) in the following subsection and focus on the numerical speedup here.

A real advantage will be obtained if the eventual number of problem instances $N_{\rm use}$ that we intend to apply the network to is sufficiently large. Specifically if $N_{\rm train}$ is the number of training samples we needed to achieve good accuracy, we have to fulfill the inequality

$$N_{\rm train}(\tau_{\rm Alg} + \tau_{\rm train}) + N_{\rm use}\tau_{\rm NN} < N_{\rm use}\tau_{\rm Alg},$$

where $\tau_{\rm Alg}$ is the run-time of the algorithm itself and $\tau_{\rm NN}$ is that of the neural network, for one problem instance. $\tau_{\rm train}$ denotes training time spent per one training sample during all epochs. If a network trained on small systems can also be applied to larger system sizes, where the algorithm run-time becomes $\tau_{\rm Alg,Large}$, we need to insert that larger value on the right-hand-side, making application of the neural network more favourable (even if its own run-time might also increase somewhat).

With this in mind, let us provide some illustrative numbers with all the caveats regarding the dependence on computer hardware and algorithm. We trained the network on small system sizes, which is inexpensive, taking just a few hours to generate $N_{\rm train} = 90,000$ samples, plus about one day to train the network. The largest system size for which we have been able to generate a few tens of instances to evaluate the neural network is $M = 22$. For this size, it takes one and a half hours to calculate the gap evolution for a single instance when applying the Lanczos algorithm. In contrast, once the neural network is trained, it is able to predict the gap evolution for this system size in 5ms. Assuming a scenario with these illustrative numbers, we can use the formula above to conclude that application of the neural network would become favourable if it were deployed on $N_{\rm use} > 20$ actual problem instances which is notably less than the number of training samples, due to the performance gains via extrapolation. Any application on more instances would yield strong time (and memory) savings in comparison to direct use of the algorithm.

In addition, besides motivating the use of a neural network via this speedup, we point out that there are also other approaches to make use of the existing training data: One could, e.g., train a neural network to solve the inverse problem, i.e. by observing the gap evolution it could try to indicate the underlying parameters in the Hamiltonian. This is otherwise a very difficult problem, for which no obvious straightforward algorithm exists, but which a neural network can learn to solve.

VII. HYBRID ALGORITHM

Our protocol can be adapted to training the neural network on larger system sizes from data generated by a quantum annealer. In this section we comment on the cost of such a hybrid implementation. We emphasize that there are a lot of caveats for such implementation, and we point out some of them in this section. Therefore, the following ideas should be taken as raw estimations.

The gap can be measured by applying different methods. Spectroscopic techniques are widely used for this purpose, for example using Ramsey-like interferometry [39, 40]. In all methods one needs to measure a time trajectory of some observable. At each time point t the experiment has to be run n times, resulting in an error that scales as $\sim 1/\sqrt{n}$, set by the projection noise. Including the number N of samples that we need to prepare for training, overall we then need nN_tN runs, where N_t stands for the number of time points. The number of training samples depends on the system size M we want to train the neural network on. Since in this work we have trained the network on sizes up to $M = 9$, it is not easy to see a clear scaling for the number of samples required to achieve a given accuracy. Therefore, we roughly estimate $N \sim 2 \times 10^5$ from our experience assuming one trains the network on the problem instances with $M = 50$. Then, taking $N_t \sim 50$ and $n \sim 10^4$, one would require around 10^{11} runs. This implies that it seems feasible to train the network on data generated from quantum annealers implemented on platforms for which each run takes up to a few microseconds, which would then require about one day to generate data to train the network. This is indeed the case for superconducting circuit platforms [41], which we propose then as our main experimental candidate.

In the future, it will be an interesting challenge to explore how much a network can deal with more noisy measurement data, reducing the need to accumulate statistics. Some efficiency improvements are possible, e.g. one might allow for a larger statistical error (smaller n) if the time points are closely spaced, because the network will then effectively try to interpolate smoothly through the noisy observations.

VIII. CONCLUSION AND OUTLOOK

In this work, we explored the power of deep learning in discovering a mapping from the parameters that fully identify a problem Hamiltonian to the full gap evolution during an adiabatic sweep. We demonstrated the CONVLSTM network succeeds in predicting the gap on the nearest neighbor connected models for which the number of parameters that identify the model scales linearly with the system size. Our CONVLSTM network is even able to predict the gap for system sizes larger than that it has been trained on and may provide some speedup in comparison with the existing exact and approximate algorithms.

While during this work we concentrated on the gap prediction in AQC, our study can provide insight for more general many-body dynamics. We conclude that a limiting factor for the learnability of such dynamics applying supervised learning is the way that parameters identifying the model scale with the system size. Our study supports the promise of CONVLSTM networks in predicting the dynamics of inhomogeneous many-body systems and their potential for extrapolating the dynamics beyond what the neural network is trained on.

Our scheme can also be applied in the context of quantum approximation optimization algorithms as it can be viewed as a trotterized version of AQC with parametrized annealing pathway. This can be explored as a future work. Overall, our scheme supports the previous efforts [3, 18] demonstrating the capability of neural networks as an instrumental tool for intuitive comprehension and rapid exploration of quantum many-body dynamics.

ACKNOWLEDGMENTS

T. B. is supported by the National Natural Science Foundation of China (62071301); State Council of the People's Republic of China (D1210036A); NSFC Research Fund for International Young Scientists (11850410426); NYU-ECNU Institute of Physics at NYU Shanghai; the Science and Technology Commission of Shanghai Municipality (19XD1423000); the China Science and Technology Exchange Center (NGA-16-001). CNB appreciates support from a Shanghai talent program and from the Shanghai Municipal Science and Technology Major Project (Grant No. 2019SHZDZX01).

Supplemental Material

In this Supplemental Material we provide a brief review of the convolutional neural network and a particular type of convolutional recurrent neural network called convolutional long short-term memory. We also provide details related to the layout of the network architectures that we applied.

I. CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) are specific type of networks with a grid-structured topology [1]. The data used with a convolutional network contain a few features at each point of the spatial grid. For example, a 2D-CNN receives an input with shape (w, h, c) where w and h represent the height and width of the spatial structure of the input, and c represents the number of features at each point of the spatial grid [1]. In our work, features at each site of the grid are identified by the coefficients that describe the spatial structure of our problem Hamiltonians as we showed in Fig. 3 (a) and Fig. 4 (e). Each convolutional layer is made of filters that are bunch of stacked kernels. Kernels are responsible to extract features of the input.

CNNs have built-in affine invariance so they can recognize patterns that are shifted or tilted in the input. One known benefit of CNNs is that they can be applied for input with varying spacial structure helping to scale up the predictions to the larger sizes. Due to this feature, they do not use the standard matrix multiplication but convolution instead. These are the main features that we exploited in this work to extrapolate the prediction of our network to the large sizes. The type of affine invariance and the locality that is required for these architectures to make best of them all exist in our nearest neighbour or the encoded version of the all-to-all connected models.

II. CONVOLUTIONAL RECURRENT NEURAL NETWORKS

In this section, we provide a brief review of the recurrent neural networks (RNNs) and a particular type of that called long short-term memory (LSTM). Then we explain an extended version of that called convolutional LSTM (CONVLSTM) network which we applied in this study.

RNNs are built of a chain of repeating modules of neural networks. Such a network introduces a feedback loop such that the output of the network at the current time depends on the current input (x_t), called the external input, and also on the perceived information from the past, called the hidden input (h_{t-1}) [37]. Such a network is able to record the history for – in principle – arbitrary long times, since weights are not time-dependent and therefore the number of trainable parameters does not grow with the time interval. For training such a network, the gradient of the cost function needs to be backpropagated from the output towards the input layer, as in feedforward

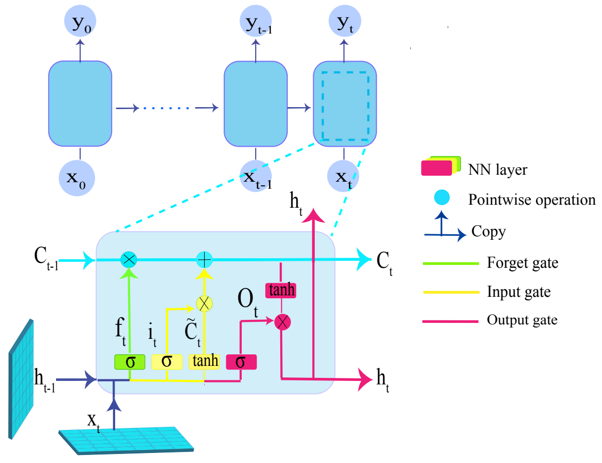

FIG. S1. An CONVLSTM network made of a chain of repeating modules, where each module includes three gates and both. Both the input-to-state and module-to-module transitions have a convolutional structure.

networks, and also along backward along the time axis.

However, RNNs are prone to run into a fundamental problem, the "vanishing/exploding gradient problem", i. e., that the size of the gradient decreases (or sometimes increases) exponentially during backpropagation. In principle, this problem can also occur in traditional feedforward networks, especially if they are deep. However, this effect is typically much stronger for RNNs since the time series can get very long. This seriously limited the trainability of early RNN architectures, which were not capable of capturing long-term dependencies. This problem led to the development of RNNs with cleverly designed gated units (controlling memory access) to avoid the exponential growth or vanishing of the gradient, and therefore permitting to train RNNs that capture both long and short-term dependencies. The first such architecture is called LSTM, developed by Hochreiter and Schmidhuber in the late 90s [42]. As an RNN architecture, standard LSTM is also built of a chain of repeating modules, as is shown in Fig. S1, where the repeating modules have a more complicated structure than in a simple recurrent network. Each module includes three gates, where each gate is composed out of a sigmoid network layer, together with the point-wise multiplication on top of it. Next, we explain step by step how these three gates together control how the memory needs to be accessed. We label weights w and biases b by subscripts according to the name of the corresponding layer.

- Forget gate layer: this gate uses the hidden state h_{t-1} from the previous time step and the external input x_t at a particular time step t (with the bias b_f and the weight w_f) to decide whether to keep the memory, or to discard the information that is of less importance, applying a sigmoid activation.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{S1}$$

  σ denotes sigmoid function and the dot stands for matrix multiplication. Eventually, the output of the forget gate is multiplied with the module state (C_t).

- Input gate layer: the operation of this gate is a three-step process,
  - first a sigmoid layer decides which data should be stored (very similar to the forget gate)

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right). \tag{S2}$$

  - hidden state and current input also will be passed into the tanh function to push values between -1 and 1 to regulate the network and stored in $\tilde{C}_t$.

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right). \tag{S3}$$

  - the outcome of the two previous steps will be combined via multiplication operation and then this information is added to the module state $(f_t * C_{t-1})$.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \tag{S4}$$

  Here the $*$ denotes element-wise multiplication.

- Output gate layer: the operation of this gate which decides the value of the next hidden input can be decomposed into two steps,
  - run a sigmoid layer on the previous hidden state and the current input, which decides what parts of the module state are going to be carried

$$O_t = \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right). \tag{S5}$$

  - passing the module state through tanh to squish the values to be between -1 and 1, and finally multiply it by the output of the sigmoid gate so that we only pass to the next module some selected parts

$$h_t = O_t * \tanh\left(C_t\right). \tag{S6}$$

When data besides temporal structure have also spatial structure the extended version of this architecture called CONVLSTM can be applied. In this case there is convolutional structure in both the input-to-module and module-to-module transitions [36] (shown with blue sheets in Fig. S1) and the internal matrix multiplications are exchanged with the convolution operations.

III. NEURAL NETWORKS LAYOUT

In this section we present the layout of the architectures that we applied for the gap prediction task during the adiabatic sweep. We have specified and trained all these different architectures with Keras [43], a deep-learning framework written for Python.

A. Fully connected neural network

In Table. I, we summarize the details related to the layout of our fully connected neural network (FCNN) for different system sizes and all the models that we explored. The training set size for all models and system sizes is 90,000. Since for the all-to-all connected model, the number of samples required for training explodes with the system size, network fails in predicting the gap for $M > 7$ using 90,000 samples for training.

The activation function for all the layers except the last layer is the rectified linear activation function ("ReLU"). For the last layer the activation function is "linear". As optimizer we always use "adam".

FCNN	All-to-All		NN		LHZ	
System size	# HL	#N/L	# HL	#N/L	# HL	#N/L
M=5	5	500	5	500	3	500
M=6	7	700	5	500	4	500
M=7	Fails		6	700	-	-
M=8	Fails		6	700	-	-
M=9	Fails		6	700	-	-

TABLE I. The layout of the FCNN for different system sizes. # HL and # N/L denote the number of hidden layers and the number of neurons per layer, respectively.

B. Convolutional long short-term memory

In this section, we present the layout of the 1D and 2D CONVLSTM architectures that we applied in the main text.

1D-CONVLSTM In Table. II, we present the layout of our 1D-CONVLSTM network applied in Sec. III B. CONVLSTM layers capture the temporal-spatial dependencies of the input. TimeDistributed is a wrapper that applies a layer to every temporal slice of an input. We use this wrapper together with the global max pooling and dense layers to transfer the input with temporal-spatial structure to the output with temporal structure.

Dense layers kill the translational invariance, therefore making the network fail to extrapolate the predictions to larger sizes. To overcome this problem, we did a pre-processing of the input data, as we explain next. Assume that we want to train the network on sizes $M \in [3, M_T]$ with N training samples for each system size, finally evaluating it on $M \in [3, M_E]$, with $M_E > M_T$ the largest extrapolation we explore. We prepare a four-dimensional array with size $N(M_T - 3) \times N_t \times M_E \times 3$ filled with zeros. Here $N(M_T - 3)$ is the total number of training samples, N_t denotes the number of time steps, and 3 denotes the number of features as we explained in the main text. Now for each sample and time step we place the string with M_T coupling coefficients randomly within the M_E zeros in this third dimension of the array. We found out that this helps the network to succeed in extrapolating.

Layers	Filters	Kernel size	Activation function
CONVLSTM1D	20	3	-
CONVLSTM1D	40	3	-
CONVLSTM1D	60	3	-
CONVLSTM1D	40	3	-
CONVLSTM1D	20	3	-
TimeDistributed(Global max pooling)			-
TimeDistributed(Dense(100)))			linear
TimeDistributed(Dense(1))			linear

TABLE II. The layout of the 2D-CONVLSTM

Note that in a CNN by default, a filter starts at the left of the input with the left-hand side of the filter placed on the far left pixels of the input. The filter is then stepped across the input until the right-hand side of the filter is placed on the far right pixels of the image. This means, the edge of the input are only exposed to the edge of the filter. However, starting the filter outside the frame of the image can give the pixels on the border of the image more of an opportunity for interacting with the filter and therefore more of an opportunity for features to be detected by the filter, and in turn, an output feature map that has the same shape as the input image. This needs to add the border of input some pixels which is called padding. But to make it clear for the network where exactly the edge of the input starts we add a row of features which is filled with ones for the range of pixels that input starts and ends identifying where a qubit exists and where not.

2D-CONVLSTM In Table. III, we present the layout of our 2D-CONVLSTM network applied in Sec. III C. We prepare the input of this network also with the same pre-processing instruction that we explained for our 1D-CONVLSTM.

Layers	Filters	Kernel size	Activation function
CONVLSTM2D	20	(2,2)	-
CONVLSTM2D	40	(2,2)	-
CONVLSTM2D	60	(2,2)	
CONVLSTM2D	40	(2,2)	-
CONVLSTM2D	20	(2,2)	-
TimeDistributed(Global max pooling)			-
TimeDistributed(Dense(100)))			linear
TimeDistributed(Dense(1))			linear

TABLE III. The layout of the 2D-CONVLSTM applied in Sec. III C.

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning* (MIT press, 2016).

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," nature **521**, 436–444 (2015).

[3] Juan Carrasquilla and Giacomo Torlai, "Neural networks in quantum many-body physics: a hands-on tutorial," arXiv preprint arXiv:2101.11099 (2021).

[4] Evert PL Van Nieuwenburg, Ye-Hua Liu, and Sebastian D Huber, "Learning phase transitions by confusion," Nature Physics **13**, 435–439 (2017).

[5] Lei Wang, "Discovering phase transitions with unsupervised learning," Phys. Rev. B **94**, 195105 (2016).

[6] Sebastian J Wetzel, "Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders," Physical Review E **96**, 022140 (2017).

[7] Juan Carrasquilla and Roger G Melko, "Machine learning phases of matter," Nature Physics **13**, 431–434 (2017).

[8] Matthew JS Beach, Anna Golubeva, and Roger G Melko, "Machine learning vortices at the kosterlitz-thouless transition," Physical Review B **97**, 045207 (2018).

[9] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo, "Neural-network quantum state tomography," Nature Physics **14**, 447–450 (2018).

[10] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac, "Neural-network quantum states, string-bond states, and chiral topological states," Phys. Rev. X **8**, 011006 (2018).

[11] Li Huang and Lei Wang, "Accelerated monte carlo simulations with restricted boltzmann machines," Phys. Rev. B **95**, 035105 (2017).

[12] Troels Arnfred Bojesen, "Policy-guided monte carlo: Reinforcement-learning markov chain dynamics," Physical Review E **98**, 063303 (2018).

[13] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh, "Accelerating quantum approximate optimization algorithm using machine learning," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2020) pp. 686–689.

[14] Matija Medvidović and Giuseppe Carleo, "Classical variational simulation of the quantum approximate optimization algorithm," npj Quantum Information **7**, 1–7 (2021).

[15] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta, "Reinforcement learning in different phases of quantum control," Phys. Rev. X **8**, 031086 (2018).

[16] Giuseppe Carleo and Matthias Troyer, "Solving the quantum many-body problem with artificial neural networks," Science **355**, 602–606 (2017).

[17] Markus Schmitt and Markus Heyl, "Quantum many-body dynamics in two dimensions with artificial neural networks," Phys. Rev. Lett. **125**, 100503 (2020).

[18] Naeimeh Mohseni, Thomas Fösel, Lingzhen Guo, Carlos Navarrete-Benlloch, and Florian Marquardt, "Deep learning of quantum many-body dynamics via random driving," (2021), arXiv:2105.00352 [quant-ph].

[19] Hsin-Yuan Huang, Richard Kueng, Giacomo Torlai, Victor V Albert, and John Preskill, "Provably efficient machine learning for quantum many-body problems," arXiv preprint arXiv:2106.12627 (2021).

[20] Filippo Vicentini, "Machine learning toolbox for quantum many body physics," Nature Reviews Physics **3**, 156–156 (2021).

[21] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser, "Quantum computation by adiabatic evolution," arXiv preprint quant-ph/0001106 (2000).

[22] Jérémie Roland and Nicolas J Cerf, "Quantum search by local adiabatic evolution," Physical Review A **65**, 042308 (2002).

[23] Daniel A Lidar, Ali T Rezakhani, and Alioscia Hamma, "Adiabatic approximation with exponential accuracy for many-body systems and quantum computation," Journal of Mathematical Physics **50**, 102106 (2009).

[24] Dorit Aharonov and Amnon Ta-Shma, "Adiabatic quantum state generation and statistical zero knowledge," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing* (2003) pp. 20–29.

[25] MHS Amin and V Choi, "First-order quantum phase transition in adiabatic quantum computation," Physical Review A **80**, 062326 (2009).

[26] Gernot Schaller, "Adiabatic preparation without quantum phase transitions," Phys. Rev. A **78**, 032328 (2008).

[27] Ralf Schützhold, "Dynamical quantum phase transitions," Journal of Low Temperature Physics **153**, 228–243 (2008).

[28] Marko Žnidarič and Martin Horvat, "Exponential complexity of an adiabatic algorithm for an np-complete problem," Phys. Rev. A **73**, 022329 (2006).

[29] A. P. Young, S. Knysh, and V. N. Smelyanskiy, "Size dependence of the minimum excitation gap in the quantum adiabatic algorithm," Phys. Rev. Lett. **101**, 170503 (2008).

[30] Naeimeh Mohseni, Marek Narozniak, Alexey N Pyrkov, Valentin Ivannikov, Jonathan P Dowling, and Tim Byrnes, "Error suppression in adiabatic quantum computing with qubit ensembles," npj Quantum Information **7**, 1–10 (2021).

[31] Murphy Yuezhen Niu, Andrew M Dai, Li Li, Augustus Odena, Zhengli Zhao, Vadim Smelyanskyi, Hartmut Neven, and Sergio Boixo, "Learnability and complexity of quantum samples," arXiv preprint arXiv:2010.11983 (2020).

[32] Hsin-Yuan Huang, Richard Kueng, and John Preskill, "Information-theoretic bounds on quantum advantage in machine learning," Phys. Rev. Lett. **126**, 190505 (2021).

[33] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing* (Ieee, 2013) pp. 6645–6649.

[34] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association* (2010).

[35] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems* (2015) pp. 802–810.

[36] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo, "Convolutional lstm network: A machine learning approach for pre-

cipitation nowcasting," arXiv preprint arXiv:1506.04214 (2015).

[37] Michael A Nielsen, *Neural networks and deep learning*, Vol. 2018 (Determination press San Francisco, CA, 2015).

[38] Wolfgang Lechner, Philipp Hauke, and Peter Zoller, "A quantum annealing architecture with all-to-all connectivity from local interactions," Science advances **1**, e1500838 (2015).

[39] Yuichiro Matsuzaki, Hideaki Hakoshima, Kenji Sugisaki, Yuya Seki, and Shiro Kawabata, "Direct estimation of the energy gap between the ground state and excited state with quantum annealing," Japanese Journal of Applied Physics **60**, SBBI02 (2021).

[40] Antonio E Russo, Kenneth Michael Rudinger, Benjamin CA Morrison, and Andrew David Baczewski, "Evaluating energy differences on a quantum computer with robust phase estimation," arXiv preprint arXiv:2007.08697 (2020).

[41] Tameem Albash and Daniel A Lidar, "Demonstration of a scaling advantage for a quantum annealer over simulated annealing," Physical Review X **8**, 031016 (2018).

[42] S Hochreiter and J Schmidhuber, "Long short-term memory. neural computation, vol. 9,№ 8, 1997, pp," (1735).

[43] François Chollet *et al.*, "Keras," `https://keras.io` (2015).