Using Shadows to Learn Ground State Properties of Quantum Hamiltonians

Viet T. Tran [1] **Laura Lewis** [2] **Hsin-Yuan Huang** [2]
Johannes Kofler [1] **Richard Kueng** [3] **Sepp Hochreiter** [1,4] **Sebastian Lehner** [1,4]

[1] ELLIS Unit Linz and LIT AI Lab, Institute for Machine Learning,
Johannes Kepler University, Linz, Austria
[2] Institute for Quantum Information and Matter,
Department of Computing and Mathematical Sciences,
Caltech, Pasadena, CA, USA
[3] Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria
[4] Institute of Advanced Research in Artificial Intelligence (IARAI), Vienna, Austria

Abstract

Predicting properties of the ground state of a given quantum Hamiltonian is an important task central to various fields of science. Recent theoretical results show that for this task learning algorithms enjoy an advantage over non-learning algorithms for a wide range of important Hamiltonians. This work investigates whether the graph structure of these Hamiltonians can be leveraged for the design of sample efficient machine learning models. We demonstrate that corresponding Graph Neural Networks do indeed exhibit superior sample efficiency. Our results provide guidance in the design of machine learning models that learn on experimental data from near-term quantum devices.

The computational investigation of quantum many-body systems is a key challenge in numerous areas of chemistry, material science, as well as of condensed-matter and high-energy physics. The underlying problem is that a complete description of such quantum systems requires in general exponentially many classical parameters. For instance, representing an arbitrary state of a couple of dozen binary quantum degrees of freedom (qubits) would exceed by far the memory capacity of presently available supercomputers [1]. Quantum computers, in contrast, do not require this quantum-to-classical conversion of information and, therefore, do not face this curse of dimensionality. However, fault-tolerant scalable quantum computers are presently out of experimental reach. Consequently, the exploration of new approaches towards an efficient representation of quantum many-body systems on classical computers is a highly active field of research.

Interestingly, it can be shown that numerous practically relevant properties of arbitrary quantum states can be captured already with informationally incomplete measurements, so called shadows [2; 3; 4]. The information to construct these shadow representations may result either from simulated or experimental measurements. Rigorous theoretical analysis shows that the number of state copies and measurements necessary to obtain classical shadow representations of a certain quality depends only logarithmically on the number of qubits n and may thus be feasible for many problems of interest [3]. Thus, classical shadows are an attractive representation of quantum states to address quantum many-body problems on classical computers.

Machine Learning (ML) proved to be a highly powerful tool-set when approaching tasks related to high dimensional data sets including a growing number of applications in the context of quantum physics [5]. The advantage of ML algorithms that are trained with classical shadow data over algorithms that do not learn from data is rigorously established for several practically relevant

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

classification and regression tasks in the context of quantum many-body systems [4]. While this result formally motivates the application of ML, it does not specify which particular ML method one should select for a given task. A relevant consideration in this context is that obtaining classical shadow data from simulations or from experiments is typically expensive and, therefore, the sample efficiency of ML algorithms is of crucial importance.
An example for a task with proven ML superiority is to predict correlations of the ground state of a certain family of quantum Hamiltonians that are defined on lattices (see Sec. 1). The graph structure inherent to this task motivates the application of Graph Neural Networks (GNNs). This type of neural network architecture is a natural choice to represent systems that are governed by pairwise interactions of components [6]. Consequently, it is not surprising that GNNs are a highly popular choice in the context of physics-related ML applications, e.g., classical mechanics [7; 8], high-energy physics [9], quantum chemistry [10], and quantum many-body physics [11; 12].

1 Problem Setting

The task considered in this work is to estimate properties of an unknown quantum many-body state which is represented by a density matrix ρ. This prediction is based on a description of a Hamiltonian for which ρ is a ground state.
The ground truth, i.e. the properties of ρ, can be efficiently estimated in a real-world experiment by the shadow measurement protocol proposed in [3]. In this protocol, a set of T copies of states is prepared and a single-qubit measurement is performed with one of the three Pauli operators X, Y or Z. The measurement basis is selected randomly for each qubit in each of the T copies of the state ρ. The selected Pauli measurements basis for each qubit along with the corresponding outcomes allow the construction of an approximation of ρ, called the classical shadow, based on which important properties of ρ can be computationally estimated in an efficient way.
In the concrete problem setting considered here, ρ is the ground state of an n-qubit 2D antiferromagnetic Heisenberg Hamiltonian. In this system, the qubits are located on a 2D lattice and adjacent qubits are coupled via an interaction term $\propto J_{ij}C_{ij}$ with uniformly sampled couplings J_{ij} from the interval $[0, 2]$ (see Fig. 1, left). The two-point correlator is defined via $C_{ij} = \frac{1}{3}(X_iX_j + Y_iY_j + Z_iZ_j)$, where the indices of the single qubit Pauli operators specify on which qubit the operator acts non-trivially. The ML task is to predict the expectation values of the two-point correlators $c_{ij}(\rho) = \mathrm{Tr}[C_{ij}\rho] \in [-1, 1]$ based on the corresponding couplings J_{ij} (see Fig. 1, middle). These couplings can be regarded as an implicit representation of ρ that does not allow an efficient, i.e. polynomial-time, calculation of the target variables $c_{ij}(\rho)$.
For this problem, [4] show that algorithms which learn from the outcomes of aforementioned shadow measurements are provably superior with respect to non-learning algorithms under standard assumptions in complexity theory. In particular, they show that the amount of training data required to train a model of a certain quality in terms of the mean squared prediction error scales at most polynomially with the system size n. However, these theoretical guarantees on the sample efficiency of learning are not expected to be tight and thus the performance of different ML methods remains to be investigated experimentally and theoretically. A practically highly relevant question in this context is whether models with suitable inductive biases can achieve an increased sample efficiency on these learning problems. To investigate this question, we generate a suitable data set. First, we compute the quasi-exact ground state with Density Matrix Renormalization Group (DMRG) [13] using the library ITensor [14]. Based on this approximation, we simulate the outcome of $T = 10^3$ randomly selected Pauli measurements and use them to construct the classical shadow representation of the ground state for each Hamiltonian according to [4]. With these classical shadows of ρ, we compute estimations of $c_{ij}(\rho)$. Using the J_{ij} of a Hamiltonian as input, the ML models are trained to predict $c_{ij}(\rho)$. For this purpose, we use the mean squared error (MSE) loss:

$$\ell = \frac{1}{n^2} \sum_{i,j} \left(\hat{c}_{ij}(\rho) - c_{ij}(\rho)\right)^2, \tag{1}$$

where $\hat{c}_{ij}(\rho)$ is the model prediction (see A.3 for details). The evaluation of the trained models is carried out on a hold-out set of Hamiltonians whose couplings are randomly sampled in the same manner as for the training set. However, in the evaluation we directly use the DMRG ground state to compute the $c_{ij}(\rho)$ and compare them to the models output. In a real-world application, on a

system that is too large for classical computational methods like DMRG one would train a model using as training labels real measurement data from an experimental realization of ρ. By using the $T = 10^3$ simulated measurements, we mimic the label noise that would result from the same number of measurements in a real experiment.

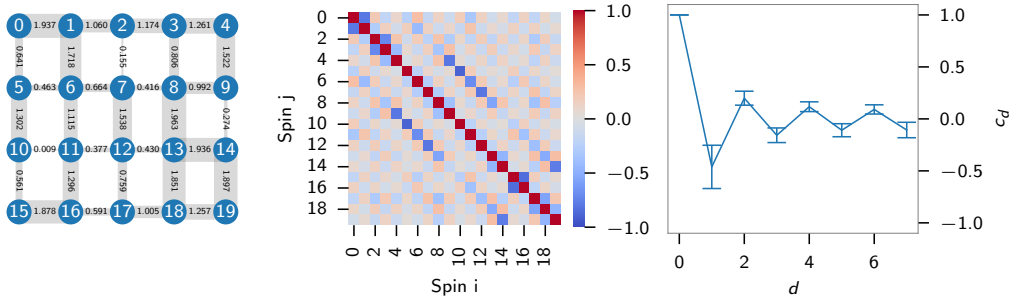

Figure 1: Left: A 2D-Heisenberg Hamiltonian system with with $n = 4 \times 5$ spins is visualized on a lattice with nodes representing different spins and with edges representing the couplings between them. The nodes are enumerated row-wise and the edge thickness indicates the coupling strength. Middle: The ground truth of the two-point correlation $c_{ij}(\rho)$ resulting from DMRG for a randomly generated Heisenberg Hamiltonian. Right: For every L_1 distance $d(i, j)$, the average coupling $c_d = \frac{1}{n_d} \sum_{i \leq j} \delta_{d(i,j),d}\, c_{ij}(\rho)$ over all possible n_d pairs with distance d is shown (δ is the Kronecker delta). Error bars indicate the standard deviation over corresponding pairs (i, j).

2 Methods

In this work, we compare three classes of ML approaches: GNNs, Multilayer Perceptrons (MLPs), and kernels.

The inputs to our learning problem can be naturally represented on a graph $G = \{\mathcal{V}, \mathcal{E}\}$ with a set of nodes $\mathcal{V} = \{v_i\},\ i \in \{1, \ldots, n\}$ which represent the n qubits and with a set of weighted edges $\mathcal{E} = \{e_{ij}\},\ i, j \in \{1, \ldots, n\}$ which represent the couplings J_{ij}. GNNs allow to build ML models that are by construction compatible with this graph structure in terms of permutation equi- and invariance and adjacency. Our GNN architectures are based on the Full GN block introduced in [6], but without global graph attributes. We compare two different variants of GNNs that are both built from several message-passing layers but which differ in the output layers (see A.2.2). One variant calculates the prediction $\hat{c}_{ij}$ directly from the corresponding edge embedding e_{ij}, i.e., $\hat{c}_{ij}(\rho) = e_{ij}$. Since we require a prediction of $c_{ij}(\rho)$ for every pair i, j, this approach necessitates an edge between every pair of nodes. We refer to it as fully-connected GNN (FC-GNN). In the second GNN variant, edges are only present for nearest-neighbor nodes, and the prediction is based on the inner product of the corresponding node-embeddings $\hat{c}_{ij}(\rho) = \langle x'_i, x'_j \rangle$. We refer to this variant as the pairwise GNN (PW-GNN). The MLP is a fully-connected neural network that takes the vectorized couplings J_{ij} as input and returns a prediction $\hat{c}_{ij}(\rho)$. Our kernel method is a reimplementation of the Neural-Tangent Kernel (NTK) method used in [4]. Details on the hyperparameter optimization, architectures and further details on the models are provided in App. A.2- A.4.

3 Experiments and Discussion

We experimentally compare various ML methods (see Sec. 2) in terms of sample efficiency in the learning problem described in Sec. 1. For each run, we train on $10, 20, 40, 80, 160$ and 320 training samples which are independently and randomly chosen from the training data set with a total of 400 Hamiltonians. Trained models are then evaluated on a test data set of 100 Hamiltonians in terms of the mean squared prediction error on the test data set MSE_{test}. In the following, error bars represent standard deviations over 10 runs with independent seeds, unless otherwise stated.

Figure 2 (left) shows the MSE_{test} for various models and different numbers N_{H} of training Hamiltonians of size $n = 4 \times 5$. In general, the GNN models perform best. For $N_{\text{H}} \geq 20$ the PW-GNN model has the lowest MSE_{test}, and for $N_{\text{H}} = 10$ the FC-GNN performs best. Next, we quantify the sample

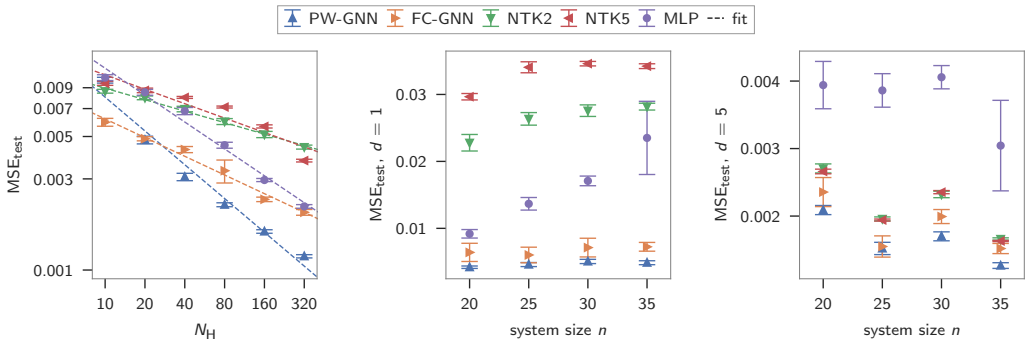

Figure 2: Left : num train vs test mse, 4x5 grid. Middle & Right : $\mathrm{MSE}_{\mathrm{test}}$ for various system sizes n with $N_{\mathrm{H}} = 80$ training Hamiltonians per system size at distance $d = 1$ (middle) and $d = 5$ (right).

efficiency of individual models. The dependence of the generalization error of ML models on the number of training samples is typically described by a power-law [15], implying $\mathrm{MSE}_{\mathrm{test}} = aN_{\mathrm{H}}^{b}$. In Figure 2 (left), power-law fits for each model are shown as dashed lines. The lower the value of the exponent b, the more sample efficient is the model. In Tab. 1, we report b for our models on various system sizes n (corresponding plots are shown in App. A.6). For each system size, the PW-GNN model is the most sample efficient, followed by the MLP. Interestingly, the most sample efficient model class PW-GNN is the only model class in these experiments that has the same graph structure as the Hamiltonians. For a given model class, the results in Tab. 1 suggest that for larger systems the sample efficiency is decreasing. When changing the system size n, several eventually counteracting effects like an exponentially growing Hilbert space or the growing information content per training sample impact the scaling behaviour of the sample efficiency. Disentangling these effects theoretically and experimentally will be the goal of future work.

System size n:	4x5	5x5	6x5	7x5
PW-GNN	**-0.59**	**-0.57**	**-0.54**	**-0.55**
FC-GNN	-0.32	-0.31	-0.29	-0.29
MLP	-0.47	-0.39	-0.36	-0.30
NTK2	-0.20	-0.18	-0.16	-0.14
NTK5	-0.25	-0.18	-0.14	-0.12

Table 1: Power-law exponents b of the fits of $\mathrm{MSE}_{\mathrm{test}}$ for different system sizes n and models. The smallest slopes are bold.

The scaling of the prediction quality with the system size is of practical interest. Since all couplings in the spin system are antiferromagnetic, the mean values of $c_{ij}(\rho)$ depend on the corresponding distances d in an alternating manner (Fig 1, right). Therefore, we analyze the prediction quality in dependence of d. We find that for $d = 1$, the $\mathrm{MSE}_{\mathrm{test}}$ is almost independent of the system size n for the GNN models and increases with n for the other models (Fig. 2, middle). For $d = 5$ all models even appear to benefit from a larger n (Fig. 2, right). Qualitatively, we observe similar trends for other training set sizes N_{H} (App. A.7). These findings are compatible with the theoretical result of [4] stating that the amount of training Hamiltonians N_{H} required to achieve a certain MSE scales at most linearly with the system size n.

4 Conclusion and Outlook

We evaluate various ML models on the task of predicting ground state properties given a description of the associated Hamiltonian. GNNs with the same graph structure as the Hamiltonians are found to perform best, in particular with respect to sample efficiency. While these results are limited to one specific class of Hamiltonians, they motivate further theoretical and experimental investigations of inductive biases of ML models in learning tasks related to graph structured Hamiltonians. The

empirically found system size dependence of the prediction quality can be regarded as encouraging for the application of classical shadows along with ML in larger experimental setups on near-term quantum devices.

5 Broader Impact

The development of sample efficient machine learning methods for learning tasks based on measurement data of quantum experiments would significantly reduce the effort required to experimentally study quantum many-body systems. Our results demonstrate that structural inductive biases can be leveraged to approach this goal and motivate broader theoretical and experimental investigations. Corresponding methods would be highly beneficial in fields involving quantum many-body systems such as e.g. chemistry and material science.

Acknowledgments and Disclosure of Funding

The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. IARAI is supported by Here Technologies. We thank the projects AI-MOTION (LIT-2018-6-YOU-212), AI-SNN (LIT-2018-6-YOU-214), DeepFlood (LIT-2019-8-YOU-213), Medical Cognitive Computing Center (MC3), INCONTROL-RL (FFG-881064), PRIMAL (FFG-873979), S3AI (FFG-872172), DL for GranularFlow (FFG-871302), AIRI FG 9-N (FWF-36284, FWF-36235), ELISE (H2020-ICT-2019-3 ID: 951847). We thank Audi.JKU Deep Learning Center, TGW LOGISTICS GROUP GMBH, Silicon Austria Labs (SAL), FILL Gesellschaft mbH, Anyline GmbH, Google, ZF Friedrichshafen AG, Robert Bosch GmbH, UCB Biopharma SRL, Merck Healthcare KGaA, Verbund AG, Software Competence Center Hagenberg GmbH, TÜV Austria, Frauscher Sensonic and the NVIDIA Corporation.

References

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[2] Scott Aaronson. Shadow tomography of quantum states. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 325–338, 2018.

[3] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

[4] Hsin-Yuan Huang, Richard Kueng, Giacomo Torlai, Victor V. Albert, and John Preskill. Provably efficient machine learning for quantum many-body problems. *Science*, 377(6613): eabk3333, 2022. doi: 10.1126/science.abk3333. URL `https://www.science.org/doi/abs/10.1126/science.abk3333`.

[5] Anna Dawid, Julian Arnold, Borja Requena, Alexander Gresch, Marcin Płodzień, Kaelan Donatella, Kim Nicoli, Paolo Stornati, Rouven Koch, Miriam Büttner, et al. Modern applications of machine learning in quantum sciences. *arXiv preprint arXiv:2204.04198*, 2022.

[6] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[7] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[8] Andreas Mayr, Sebastian Lehner, Arno Mayrhofer, Christoph Kloss, Sepp Hochreiter, and Johannes Brandstetter. Boundary graph neural networks for 3d simulations. *arXiv preprint arXiv:2106.11299*, 2021.

[9] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[10] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1): 1–11, 2022.

[11] Dmitrii Kochkov, Tobias Pfaff, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Bryan K Clark. Learning ground states of quantum hamiltonians with graph networks. *arXiv preprint arXiv:2110.06390*, 2021.

[12] Li Yang, Wenjun Hu, and Li Li. Scalable variational monte carlo with graph neural ansatz. *arXiv preprint arXiv:2011.12453*, 2020.

[13] Steven R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, Nov 1992. doi: 10.1103/PhysRevLett.69.2863. URL `https://link.aps.org/doi/10.1103/PhysRevLett.69.2863`.

[14] Matthew Fishman, Steven R. White, and E. Miles Stoudenmire. The ITensor software library for tensor network calculations, 2020.

[15] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[17] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020. URL `https://github.com/google/neural-tangents`.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We support our claims in Section 3.
   (b) Did you describe the limitations of your work? [Yes] In the Section 4.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We did not yet include code because this work is in progress. However, a public code repository is under preparation.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specify all data splits and hyperparameters in A.3.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report the standard deviation of the MSE_{test} for 10 different seeds as can be seen in Fig. 2.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We state compute resources in A.5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] The original data set is provided in [4] as mentioned in A.4

(b) Did you mention the license of the assets? [Yes] It is open source: `https://github.com/hsinyuan-huang/provable-ml-quantum`

(c) Did you include any new assets either in the supplemental material or as a URL? [No] The additional data set will be published along with the public code repository which is under preparation.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Structure of Input Data

For the GNN models, the 2D-Heisenberg systems were encoded into a graph structure in which the nodes are represented as one-hot vectors enumerated as shown in Fig. 3. For FC-GNNs, we connect all nodes and include self-connections. We initialize the edge weights with the corresponding couplings J_{ij}. In the cases where the nodes are not neighbors and thus actually do not directly interact in the Heisenberg system, we initialize the corresponding edge weights to 0. For the PW-GNNs, we only connect neighboring nodes. The edge weights are initialized with the corresponding couplings J_{ij} and the nodes are one-hot encoded as well. The MLP models and kernel methods take as input the vectorized couplings J_{ij} where the edges are vectorized according to the enumeration in Fig. 3.

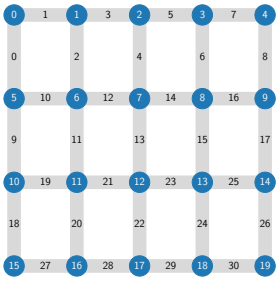

Figure 3: Enumeration order of edges and nodes for a 4×5 grid

A.2 Model Architecture

A.2.1 MLP Model

Given an $m \times 5$-grid Heisenberg Hamiltonian H with $n = 5m$ qubits with n^2 two-point correlations with vectorized coupling $x \in [0, 2]^{5(m-1)+4m}$ the MLP model first projects x into an h-dimensional representation.

Next, we stack multiple MLP layers, enumerated by l and keep the hidden dimensionality constant. After the last MLP layer, we use a linear layer to project the hidden representation to the output size of n^2 and normalize it. We denote the result as z. Finally, we reshape z into a matrix $Z \in \mathbb{R}^{n \times n}$ and predict all two-point correlations $c_{ij}(\rho)$ of the Hamiltonian H by constructing the following symmetric matrix:

$$\hat{c}_{ij}(\rho) = (ZZ^T)_{ij}. \tag{2}$$

Thus, the MLP model consists of l MLP layers and a final linear layer, followed by a instance-wise normalization and symmetrization operation.

A.2.2 GNN Models

Given the same $m \times 5$ grid Heisenberg Hamiltonian H, our input data now consists of graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$, $|\mathcal{V}| = m$ and edges $e_{ij} \in \mathcal{E}$, $|\mathcal{E}| = n_\text{e}$. We assign each node a positional embedding by attaching a one-hot encoded vector as a d_p dimensional node feature p_{v_i}. Each edge is assigned a scalar edge feature $a_{ij} \in [0, 2]$ storing the corresponding coupling J_{ij} for this edge.
Here d_p denotes the dimension needed to encode all nodes as one-hot vectors and thus $d_\text{p} = n$ and n_e denotes the number of edges which differ for PW-GNN and FC-GNN:

$$n_\text{e} = \begin{cases} (5m)^2 & \text{for FC-GNN,} \\ 5(m-1) + 4m & \text{for PW-GNN.} \end{cases} \tag{3}$$

We use message passing neural networks (MPNN) [16; 6] as a specific type of graph neural networks to operate on nodes and edges in the hidden dimension h. In an MPNN the edge embeddings g_{ij} at edge e_{ij} and node embeddings h_i at node v_i are iteratively updated via

$$m_{ij} = \phi(h_i, h_j, g_{ij}) \tag{4}$$
$$g'_{ij} = \omega(g_{ij}, m_{ij}) \tag{5}$$
$$h'_i = \psi\left(h_i, \square_{e_{ij} \in \mathcal{E}} m_{ij}\right), \tag{6}$$

where $\square_{e_{ij} \in \mathcal{E}}$ denotes the aggregation of messages m_{ij} over all edges e_{ij} with a permutation invariant operation such as mean, max or sum. We use summation by default. The functions ϕ, ψ and ω are trainable and unique for each MPNN layer. In our case they are d_MLP-layer MLPs with ReLU activations and hidden dimension h.

Our GNN models consist of an initial MPNN layer MPNN_proj which projects the input node embeddings $\{p_{v_i}\}$ and edge embeddings $\{a_{ij}\}$ into an h-dimensional hidden representation by ψ, ϕ and ω via the equations (4)-(6):

$$\{h_i^{(1)}\}, \{m_{ij}^{(1)}\} = \text{MPNN}_\text{proj}\left(\{p_{v_i}\}, \{a_{ij}\}\right). \tag{7}$$

Thus, $h_i^{(1)} \in \mathbb{R}^h$ and $m_{ij}^{(1)} \in \mathbb{R}^h$. For a d_MPNN-layer GNN model, we now apply $d_\text{MPNN} - 2$ layers of MPNNs with skip connections whose update procedure is defined as:

$$m_{ij} = \phi(h_i, h_j, g_{ij}) \tag{8}$$
$$g'_{ij} = \omega(g_{ij}, m_{ij}) + g_{ij} \tag{9}$$
$$h'_i = \psi\left(h_i, \square_{e_{ij} \in \mathcal{E}} m_{ij}\right) + h_i. \tag{10}$$

For this step, the old and new node and edge embeddings have the same dimension, i.e. $h_i, g_{ij}, h'_i, g'_{ij}, m_{ij} \in \mathbb{R}^h$. The final readout layers differ for the PW-GNN and FC-GNN.
In case of the FC-GNNs the final readout layer is an MPNN without skip connections, which projects the node embeddings down to $\mathbb{R}^n$ and the edge embeddings down to a scalar value. Those scalar edge embeddings $\{g_{ij}\}_{i,j=1}^n$ are reshaped into a matrix $Z \in \mathbb{R}^{n \times n}$ and then multiplied with its transpose to get a symmetric two-point correlation matrix. In order to confine the correlation predictions to $[-1, +1]$ we apply an element-wise tanh operation:

$$\hat{c}_{ij}(\rho) = (\tanh\left(ZZ^T\right))_{ij}. \tag{11}$$

A possible alternative which we will test in the future would be similar to the MLP method where we apply a normalization before symmetrization and leave out the $\tanh$ activation on the output.
For PW-GNNs the final readout layer is an MPNN-layer, however, with skip connections and thus the final node embeddings $\{h_i\}_{i=1}^n$ and edge embeddings $\{g_{ij}\}_{i,j=1}^n$ are still h-dimensional. We normalize each $\tilde{h}_i = h_i / \|h_i\|_2$ and calculate the $\hat{c}_{ij}(\rho)$ of the Hamiltonian H as

$$\hat{c}_{ij}(\rho) = \langle \tilde{h}_i, \tilde{h}_j \rangle. \tag{12}$$

In both GNN-model types we have in total d_{MPNN} MPNN layers where the first one is a projection MPNN and the hidden MPNN layers operate on h-dimensional node and edge embeddings. The last layer for FC-GNNs is a down projection MPNN so that we obtain a scalar edge embedding. Since for PW-GNNs the correlation are estimated via a dot-product of node embeddings we do not apply a down projection and proceed with a final MPNN operating in the h-dimensional embedding space.

A.2.3 Kernel Based Models

We reimplemented the neural tangent kernel of a 2-layer (NTK2) and 5-layer (NTK5) fully connected network with ReLU activations and an instance normalization as the first layer similar to the architecture proposed in [4] using the Neural Tangents library [17].
Let us denote the neural tangent kernel of an l-layer fully connected network as $\kappa(x_l, x_m)$, with x_l, x_m being the vectorized couplings of Hamiltonians H_l and H_m. Then, we calculate the instance-wise normalized feature vector ϕ_l for the l-th sample as $\tilde{\phi}_l = (\kappa(x_l, x_1), \ldots, \kappa(x_l, x_{N_\text{H}}))$ and $\phi_l = \tilde{\phi}_l / \|\tilde{\phi}_l\|_2$. We can now write the full feature matrix of all training samples as $\Phi \in \mathbb{R}^{N_\text{H} \times N_\text{H}}$ where each row l corresponds to the feature vector of H_l. Given those feature vectors, we train two types of linear regression models with different hyperparameters using scikit-learn [18].
The linear kernel ridge regression for the two-point correlation $c_{ij}(\rho)$ then solves the following problem individually for each ij-pair of nodes:

$$\min_w ||\Phi w - c_{ij}(\rho)||_2^2 + \alpha ||w||_2^2 \tag{13}$$

Here, w are a trainable model parameters and α are different regularization strengths which are found via a hyperparameter search (see App. A.4). The linear support vector regression solves the following optimization problem for each ij-pair of nodes, where $c_{ij}(\rho_l)$ denotes the correlations for the ground state ρ_l corresponding to the Hamiltonian H_l:

$$\min_{w,b} \frac{1}{2} w^T w + K \sum_{l=1}^{N_\text{H}} \max(0, |c_{ij}(\rho_l) - (w^T \Phi_l + b)| - \varepsilon) \tag{14}$$

Here, w and b denote trainable model parameters, Φ_l is the l-th column of the full feature matrix Φ and K is a regularization parameter determined by a hyperparameter search (see App. A.4) and as in the original work [4] we set $\varepsilon = 0.1$.

A.3 Training and Validation

For the experiments shown in Sec. 3, we use a fixed train/validation/test data set split of 400/100/100 for all experiments. The splits are randomly generated once before all experiments. The actually utilized training data sets of size $N_\text{H} \in \{10, 20, 40, 80, 160, 320\}$ is randomly sampled from the total 400 training Hamiltonians. Each error bar displayed in Fig. 2 is calculated as the standard deviation over 10 different seeds $\{s_i = 7k | k = 0, \ldots, 9\}$ which affect the model initialization weights and the composition of the training data set. For each data set size N_H, the same 10 seeds are used so that for the same seed the smaller training data sets are subsets of the larger training data sets. More formally, let $\mathcal{T}_{\text{train}}(s_i, N_\text{H})$ denote the training data set for seed s_i with N_H samples. Then, for any fixed s_i, we have $\mathcal{T}_{\text{train}}(s_i, 10) \subset \mathcal{T}_{\text{train}}(s_i, 20) \subset \cdots \subset \mathcal{T}_{\text{train}}(s_i, 320)$. In total we have 600 Hamiltonians for each grid size $\{4, 5, 6, 7\} \times 5$ with identical splitting compositions for each of the 10 seeds.

The validation data set is used to evaluate the models after every epoch to apply early stopping and a learning rate schedule. This is only valid for the MLP and GNN models since the kernel methods use another form of training which will be explained in section A.3.1. More specifically, after each training epoch we compare the validation MSE with the validation MSE of the previous epoch and stop the training if the MSE did not decrease for 100 epochs. Furthermore, we decrease the learning rate by a factor of 0.9 if the validation MSE did not decrease for 25 epochs. If nothing else is stated, the initial learning rate is 10^{-3} and the default batch size is 10.

A.3.1 Training and Validation Details of Kernel Methods

We deviate from the original framework (see [4]) insofar as we calculate the full feature matrix Φ only with respect to the training samples in order to align with the more common training method of

the other models. Thus, we operate in an inductive training setting instead of the original transductive setting. Using the instance-wise normalized full feature matrix Φ, we perform separate regressions on the two-point correlations $c_{ij}(\rho)$ for each pair of nodes for each model variant. Then, we evaluate the trained models on the validation split and select the model type and the respective hyperparameters by choosing the configuration with the lowest validation MSE.

A.4 Hyperparameter Search

Before carrying out the experiments with different data set sizes on different grid sizes, we perform various hyperparameter searches for each model to find well working configurations. Those searches are performed on 4×5 grids of the original data set provided by [4] with 100 Hamiltonians. We use a 60/40 split for training and validation. Early stopping and learning rate scheduling are used as well.

We randomly select three initial learning rates l_0 in the interval $[1 \times 10^{-4}, 5 \times 10^{-2}]$ for each combination of the other hyper parameters. For the FC-GNNs we vary the depth of the MLP layers d_{MLP} which are used in each MPNN-layer, the number of MPNN layers d_{MPNN} and the hidden size of the MLPs h. Thus the combinations of hyperparameters tested are $\{(l_0, d_{\text{MLP}}, d_{\text{MPNN}}, h) | \, l_0 \sim \mathcal{LU}[1 \times 10^{-4}, 5 \times 10^{-2}], \, d_{\text{MLP}} \in \{1, 2, 3, 4, 6\}, \, d_{\text{MPNN}} \in \{1, \ldots, 9\}, \, h \in \{10, 20, 40, 80, 160, 320\}\}$ where $\mathcal{LU}[a, b]$ denotes the log-uniform distribution over the interval $[a, b]$.

Similarly, the PW-GNNs were tested with different initial learning rates l_0, different MPNN depths d_{MPNN}, MLP depths d_{MLP} and hidden sizes of the MLPs h as well and thus the combinations of tested hyperparameters is also $\{(l_0, d_{\text{MLP}}, d_{\text{MPNN}}, h) | \, l_0 \sim \mathcal{LU}[1 \times 10^{-4}, 5 \times 10^{-2}], \, d_{\text{MLP}} \in \{1, 2, 3, 4, 6\}, \, d_{\text{MPNN}} \in \{1, \ldots, 9\}, \, h \in \{10, 20, 40, 80, 160, 320\}\}$.

The MLP models are also tested with different initial learning rates l_0, MLP depths d_{MLP} and hidden sizes h. Furthermore MLPs are also tested with different activation functions σ_{act}. Thus the tested combinations are $\{(l_0, d_{\text{MLP}}, h, \sigma_{\text{act}}) | \, l_0 \sim \mathcal{LU}[1 \times 10^{-4}, 5 \times 10^{-2}], \, d_{\text{MLP}} \in \{1, 2, 3, 4, 6\}, \, h \in \{10, 20, 40, 80, 160, 320\}, \, \sigma_{\text{act}} \in \{\text{ReLU}, \tanh, \text{SELU}\}\}$. The found configurations are summarized in the Tab. 2

	d_{MPNN}	d_{MLP}	h	$\sigma_{\text{activation}}$
PW-GNN	5	2	320	—
FC-GNN	6	3	40	—
MLP	—	2	320	tanh

Table 2: Results of the hyperparameter optimization for each model.

The FC-GNNs and MLP models are found to be insensitive to the exact learning rates as long as the order of magnitude stayed in certain ranges. Thus, we choose a learning rate of 1×10^{-3} and for PW-GNNs a smaller learning rate of 1×10^{-4} works well.

The NTK architecture is taken from [4]. The downstream regression models after the NTK are tuned in the same manner as in [4]. There, each two-point correlation $c_{ij}(\rho)$ is learned by a separate regression model. One chooses from a linear kernel ridge regression model with different regularization strengths $\alpha \coloneqq \frac{1}{2K}$ and $\alpha \in \{0.25, 0.5, 1, 2, 4, 10, 20, 40\}$ and a linear support vector regression model with different regularization strengths $K \in \{0.0125, 0.025, 0.05, 0.125, 0.25, 0.5, 1, 2\}$.

A.5 Compute Resources

For our computations we use a NVIDIA GTX 1080Ti GPU and an Intel Xeon E5-2687W v4 CPU. A graph based model typically finishes training in less than 10 minutes for smaller grid sizes. For larger grids the training time increases to about 1h. The MLP and kernel methods usually finish training in under 5 minutes for each grid size and N_{H}.

A.6 Dependence of MSE_{test} on the system size n

In the following we list the fit results on the MSE_{test} in Tab. 4 and Tab. 3 for completeness

System size n:	4x5	5x5	6x5	7x5
PW-GNN	-3.47	-3.66	-3.78	-3.93
FC-GNN	-4.34	-4.57	-4.66	-4.84
MLP	-3.40	-3.70	-3.83	-4.16
NTK2	-4.26	-4.43	-4.58	-4.80
NTK5	-3.98	-4.33	-4.55	-4.78

Table 3: Logarithm of the power-law ($\propto aN_{\mathrm{H}}^{b}$) pre-factor $a' = \log a$ of the fits of $\mathrm{MSE}_{\mathrm{test}}$ for different system sizes n and models.

System size n:	4x5	5x5
PW-GNN	$\begin{bmatrix} 0.00021 & -0.00084 \\ -0.00084 & 0.00368 \end{bmatrix}$	$\begin{bmatrix} 0.00019 & -0.00075 \\ -0.00075 & 0.00328 \end{bmatrix}$
FC-GNN	$\begin{bmatrix} 0.00008 & -0.00031 \\ -0.00031 & 0.00135 \end{bmatrix}$	$\begin{bmatrix} 0.00007 & -0.00028 \\ -0.00028 & 0.00121 \end{bmatrix}$
MLP	$\begin{bmatrix} 0.00009 & -0.00036 \\ -0.00036 & 0.00159 \end{bmatrix}$	$\begin{bmatrix} 0.00015 & -0.00062 \\ -0.00062 & 0.00271 \end{bmatrix}$
NTK2	$\begin{bmatrix} 0.00002 & -0.00007 \\ -0.00007 & 0.00029 \end{bmatrix}$	$\begin{bmatrix} 0.00002 & -0.00007 \\ -0.00007 & 0.00029 \end{bmatrix}$
NTK5	$\begin{bmatrix} 0.00014 & -0.00057 \\ -0.00057 & 0.00249 \end{bmatrix}$	$\begin{bmatrix} 0.00007 & -0.00027 \\ -0.00027 & 0.0012 \end{bmatrix}$
System size n:	6x5	7x5
PW-GNN	$\begin{bmatrix} 0.0001 & -0.00041 \\ -0.00041 & 0.0018 \end{bmatrix}$	$\begin{bmatrix} 0.00009 & -0.00037 \\ -0.00037 & 0.00162 \end{bmatrix}$
FC-GNN	$\begin{bmatrix} 0.00009 & -0.00035 \\ -0.00035 & 0.00152 \end{bmatrix}$	$\begin{bmatrix} 0.00014 & -0.00056 \\ -0.00056 & 0.00246 \end{bmatrix}$
MLP	$\begin{bmatrix} 0.00018 & -0.00072 \\ -0.00072 & 0.00314 \end{bmatrix}$	$\begin{bmatrix} 0.0002 & -0.00082 \\ -0.00082 & 0.00358 \end{bmatrix}$
NTK2	$\begin{bmatrix} 0.00001 & -0.00006 \\ -0.00006 & 0.00026 \end{bmatrix}$	$\begin{bmatrix} 0.00002 & -0.00008 \\ -0.00008 & 0.00035 \end{bmatrix}$
NTK5	$\begin{bmatrix} 0.00003 & -0.00011 \\ -0.00011 & 0.00046 \end{bmatrix}$	$\begin{bmatrix} 0.00002 & -0.00006 \\ -0.00006 & 0.00027 \end{bmatrix}$

Table 4: Covariance matrices for the power-law fit ($\propto aN_{\mathrm{H}}^{b}$) parameters b and $a' = \log a$ of the fits of $\mathrm{MSE}_{\mathrm{test}}$ for each system size n and model. The top left matrix element is the variance of parameter b

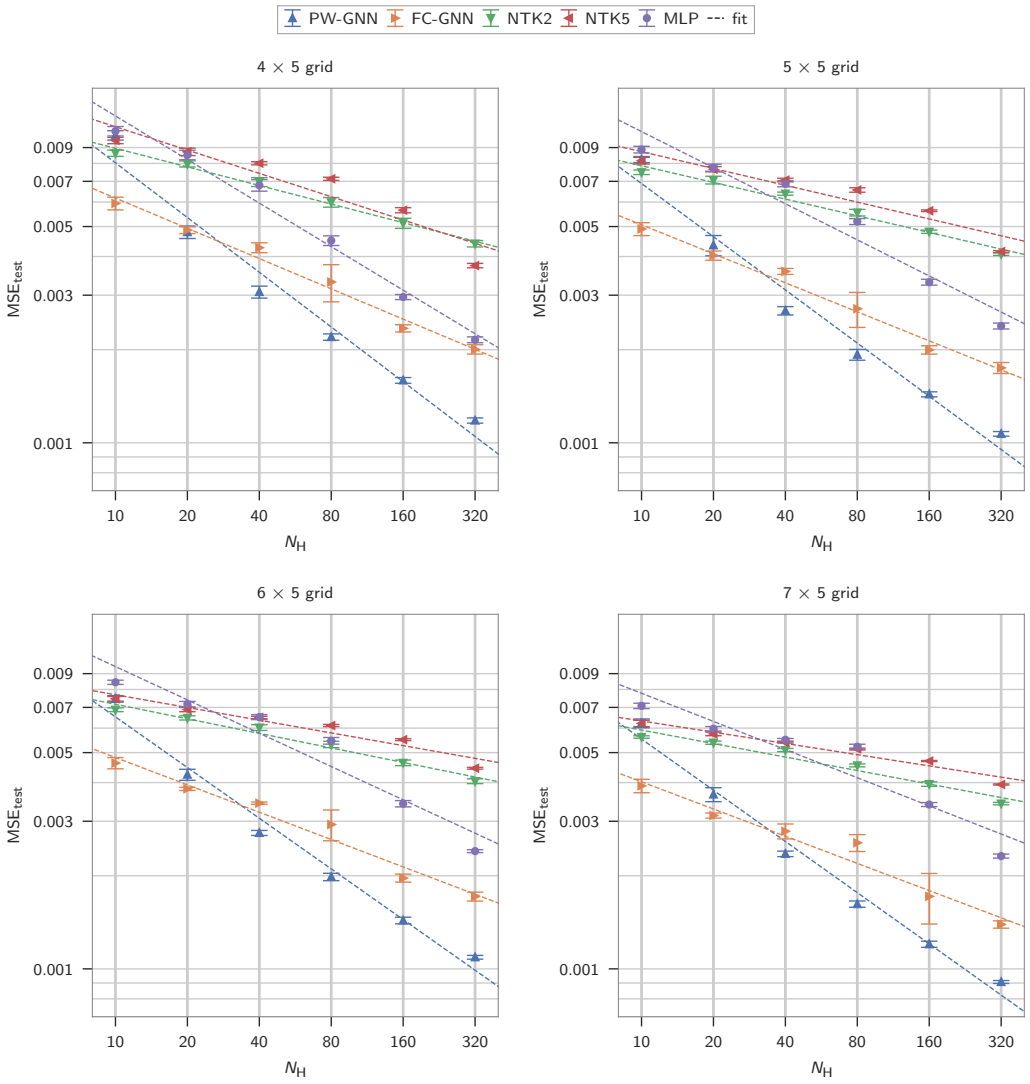

Figure 4: $\mathrm{MSE}_{\mathrm{test}}$ over different number of training data set sizes N_{H} with according power-law fits for each model and grid size.

A.7 Dependence of $\mathrm{MSE}_{\mathrm{test}}$ on the system size n for $\hat{c}_{ij}(\rho)$ with $d = 1, 3, 5$

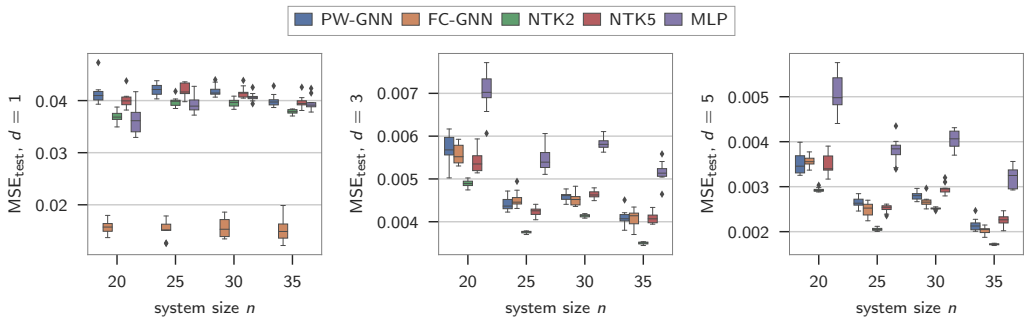

Figure 5: $\mathrm{MSE}_{\mathrm{test}}$ over system size n for distance $d = 1, 3, 5$ for $N_{\mathrm{H}} = 10$

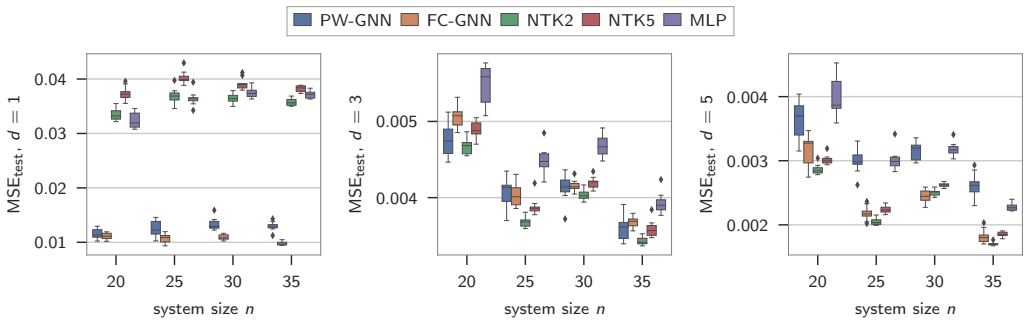

Figure 6: MSE_{test} over system size n for distance $d = 1, 3, 5$ for $N_{\text{H}} = 20$

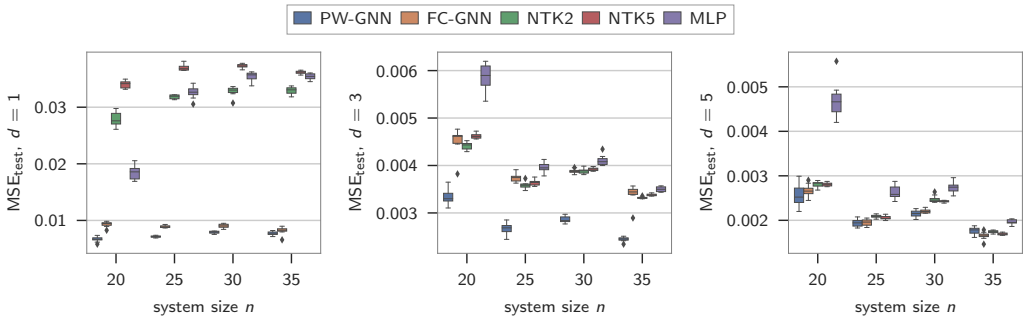

Figure 7: MSE_{test} over system size n for distance $d = 1, 3, 5$ for $N_{\text{H}} = 40$

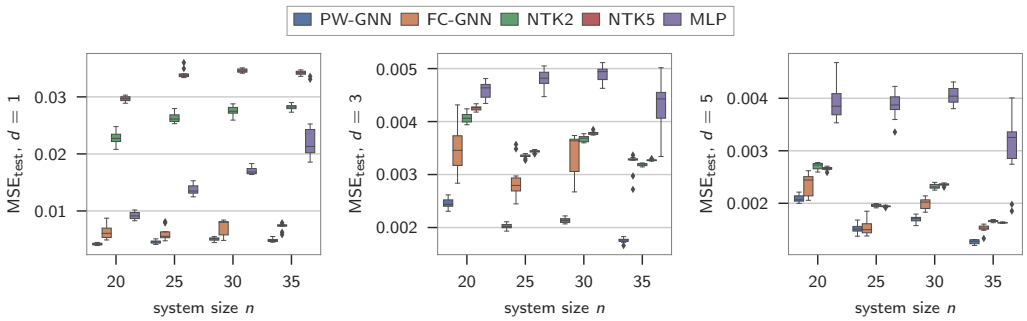

Figure 8: MSE_{test} over system size n for distance $d = 1, 3, 5$ for $N_{\text{H}} = 80$

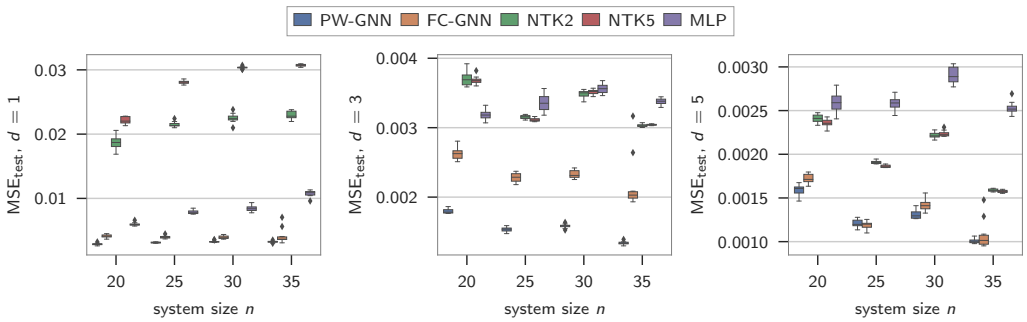

Figure 9: $\mathrm{MSE}_{\mathrm{test}}$ over system size n for distance $d = 1, 3, 5$ for $N_{\mathrm{H}} = 160$

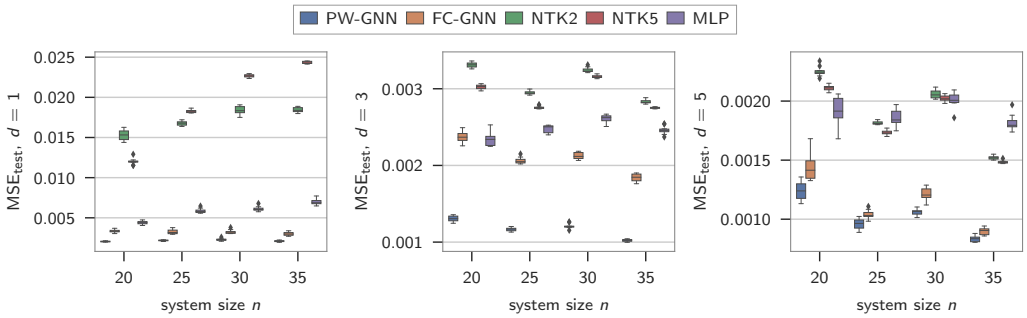

Figure 10: $\mathrm{MSE}_{\mathrm{test}}$ over system size n for distance $d = 1, 3, 5$ for $N_{\mathrm{H}} = 320$