Generalization capabilities of translationally equivariant neural networks

Srinath Bulusu,[*] Matteo Favoni,[†] Andreas Ipp,[‡] David I. Müller,[§] and Daniel Schuh[‖]
Institute for Theoretical Physics, TU Wien, Austria

(Received 7 April 2021; accepted 16 August 2021; published 6 October 2021)

The rising adoption of machine learning in high-energy physics and lattice field theory necessitates the reevaluation of common methods that are widely used in computer vision, which, when applied to problems in physics, can lead to significant drawbacks in terms of performance and generalizability. One particular example for this is the use of neural network architectures that do not reflect the underlying symmetries of the given physical problem. In this work, we focus on complex scalar field theory on a two-dimensional lattice and investigate the benefits of using group equivariant convolutional neural network architectures based on the translation group. For a meaningful comparison, we conduct a systematic search for equivariant and nonequivariant neural network architectures and apply them to various regression and classification tasks. We demonstrate that in most of these tasks our best equivariant architectures can perform and generalize significantly better than their nonequivariant counterparts, which applies not only to physical parameters beyond those represented in the training set, but also to different lattice sizes.

DOI: 10.1103/PhysRevD.104.074504

I. INTRODUCTION

Machine learning has become an increasingly popular tool for a diverse range of applications in physics. Particularly suitable for the analysis of spatially arranged data are convolutional neural networks (CNNs). Modern CNN architectures are based on the idea that a network's prediction should not change when the input is shifted. They rely on two key ingredients that have already been introduced by the neocognitron [1] over 40 years ago: convolutional layers (S cells) and pooling (subsampling, downsampling) layers (C cells). This incorporation of a translational symmetry was an essential advantage over its predecessor, the cognitron [2]. However, equivariance under translations is not guaranteed in a generic CNN, even though it is the idea behind weight sharing in the convolutional layers.

In the past decade, the computer vision community has created many different machine learning algorithms and continues to refine them. During the ImageNet large scale visual recognition challenge (ILSVRC) [3], which was a popular competition that was held annually from 2010 until 2017, the performance of CNNs steadily increased, and, in 2012, AlexNet [4] was the first CNN to win the classification task. However, its first convolutional layer already breaks translational equivariance by using a stride of four, as do three max pooling layers with a stride of two that are part of the network. Additionally, the output of the last convolutional layer is flattened before it is passed to the dense layers of the network. LeNet-5 [5], a very early CNN, uses a stride of one in the convolutional layers, but the average pooling layers with a stride of two break translational symmetry. An important step toward a translationally equivariant network architecture has been made by the introduction of global pooling layers. Global average pooling (GAP) was first introduced in Ref. [6], and the first winning network of the ILSVRC's classification task that makes use of it is ResNet [7] from 2015's competition.

The grand success of machine learning in many different tasks has also garnered attention within other research communities. Although many ingredients can be carried over from computer vision, differences in the tasks may require a different treatment. A lot of effort has been made to incorporate global [8–18] and gauge [19–21] symmetries in the network architecture, since they play a central role in modern physics, among other fields. Nevertheless, the most basic one, translational symmetry, is often not strictly enforced despite the fact that the task would allow for it. Oftentimes, the data are flattened somewhere in the network, as, e.g., in Refs. [22–27], and sometimes a convolutional or pooling operation with a stride greater than one spoils symmetry under translations, even though a global pooling layer constitutes the transition from the convolutional part of the network to its dense part, e.g., in Ref. [28].

[*]sbulusu@hep.itp.tuwien.ac.at
[†]favoni@hep.itp.tuwien.ac.at
[‡]ipp@hep.itp.tuwien.ac.at
[§]dmueller@hep.itp.tuwien.ac.at
[‖]Corresponding author.
schuh@hep.itp.tuwien.ac.at

There are examples that make explicit use of this symmetry though, such as Ref. [29], and we want to raise awareness that one should take it into account when choosing a network architecture.

In this paper, we focus on translational symmetry in CNNs. In addition to providing theoretical reasons for choosing a translationally equivariant architecture, we conduct experiments with three types of architecture on three different machine learning tasks. We try to find well-perform ing architectures for each of these types by doing an extensive search with the optimization framework OPTUNA [30]. We furthermore investigate the generalization capabilities of the three network types in two different ways. First, we examine how models generalize to different sets of physical parameters at a fixed lattice size, and, second, we inquire how well they generalize to other lattice sizes. The latter is not possible for networks including a flattening step, since they require a fixed input size. Therefore, these types of model would have to be retrained for each new lattice size.

This paper is structured as follows: We first discuss translational symmetry in Sec. II and show under which circumstances CNNs are indeed respecting equivariance under translations, as well as certain pitfalls that break the symmetry, in Sec. III. The next three sections are devoted to three different machine learning tasks pertaining to a complex scalar field on a lattice: a regression task with the aim of predicting two observables of said scalar field (Sec. IV), a classification task, in which the algorithm should judge whether or not the flux of a given lattice configuration is conserved (Sec. V), and another regression task, in which the network is supposed to figure out how many flux violations are present on the lattice (Sec. VI). Section VII contains our conclusions and possible future research avenues. The Appendixes encompass information about the complex scalar field (Appendix A), our datasets (Appendix B), some supplemental proofs for Sec. III (Appendix C), and some additional analysis pertaining to the regression task in Sec. IV (Appendix D).

II. TRANSLATIONAL SYMMETRY

In this section, we exemplify symmetry aspects on a complex scalar field and explain how they may impact the choice of machine learning models. The action of a complex scalar field ϕ in an external potential V in D dimensions can be written as

$$S = \int \mathrm{d}^D x \mathcal{L}, \tag{1}$$

with the Lagrangian density

$$\mathcal{L} = \partial_\mu \phi^* \partial^\mu \phi - V(\phi^* \phi). \tag{2}$$

The latter is covariant under translations $x^\mu \to x'^\mu = x^\mu + a^\mu$, with a constant vector a^μ. This can be seen by noting that the fields transform via $\phi'(x^\mu) = \phi(x^\mu - a^\mu)$ and that the partial derivative is not influenced by translations $\partial'_\mu = \partial_\mu$. The action given by Eq. (1) is then invariant under translations. This has important implications for the resulting physical theory, because finite continuous symmetries of the action lead to conserved quantities, according to Noether's first theorem [31]. The invariance under temporal translations entails energy conservation; the invariance under spatial translations leads to momentum conservation.

Another important symmetry of the action in Eq. (1) is a global $U(1)$ symmetry, given by $\phi \to e^{i\alpha}\phi$. It implies the existence of a conserved four-current j^μ and allows the definition of a chemical potential μ. The action can be modified to directly include the chemical potential via

$$S = \int \mathrm{d}x_0 \mathrm{d}^{D-1} x (|D_0 \phi|^2 - |\partial_i \phi|^2 - V(|\phi|)), \tag{3}$$

with $D_0 = \partial_0 - i\mu$.

In the following, we consider a complex scalar field in $1+1$ dimensions in a quartic potential

$$V(|\phi|) = m^2 |\phi|^2 + \lambda |\phi|^4 \tag{4}$$

on the lattice with periodic boundary conditions. The parameters of the potential are the mass m and the coupling constant λ. A discretized version of the action in Eq. (3) retains its invariance under discrete translations. We then switch to a dual representation, called flux representation. It describes the same physical content as the original representation, but the variables are four integer fields (link variables) $k_{x,\nu}$ and $l_{x,\nu}$, with $\nu = 1, 2$, instead of the complex scalar field. The corresponding partition function reads

$$Z = \sum_{\{k,l\}} \left(\prod_{x,\nu} \frac{1}{(|k_{x,\nu}| + l_{x,\nu})! l_{x,\nu}!} \right) \left(\prod_x e^{\mu k_{x,2}} W(f_x) \right) \times \left(\prod_x \delta \left(\sum_\nu (k_{x,\nu} - k_{x-\hat{\nu},\nu}) \right) \right), \tag{5}$$

where the outer sum is a shorthand for

$$\sum_{\{k,l\}} = \prod_{x,\nu} \sum_{k_{x,\nu}=-\infty}^{\infty} \sum_{l_{x,\nu}=0}^{\infty}. \tag{6}$$

The function $W(f_x)$ is given by

$$W(f_x) = \int_0^\infty \mathrm{d}x x^{f_x+1} \mathrm{e}^{-\eta x^2 - \lambda x^4}, \tag{7}$$

and the integer field f_x is defined as

$$f_x = \sum_\nu [|k_{x,\nu}| + |k_{x-\hat{\nu},\nu}| + 2(l_{x,\nu} + l_{x-\hat{\nu},\nu})]. \quad (8)$$

The dual formulation incorporates the same symmetry properties as the original formulation. A more detailed explanation of this procedure is given in Appendix A and in Ref. [32]. To ensure the flux conservation demanded by the Kronecker δ symbol in Eq. (5), the worm algorithm [33] has been employed to update the link variables $k_{x,\nu}$. It is a local algorithm that updates contiguous field values on the lattice in successive steps. The resulting structures are known as worms. When the head of a worm meets its tail, a worm is closed; otherwise, it is open. Details about the generation of our datasets can be found in Appendix B.

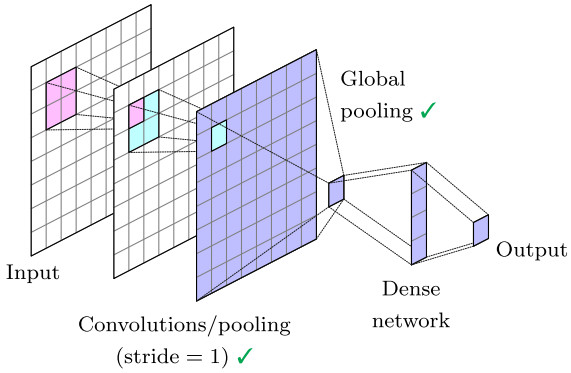
This dual representation in two dimensions allows for a strong analogy with two-dimensional images. While every pixel of an image is described by one (grayscale) or three (color) numbers, every position of the dual lattice is described by four values. An important difference between them is their boundary conditions. Typically, image applications employ fixed boundary conditions, which break translational equivariance at the boundaries. In contrast, using periodic boundary conditions on the lattice, translational equivariance can be preserved.

The aforementioned four integer fields of the flux representation are used as the input for the upcoming machine learning tasks. In these tasks, the intensive or extensive nature of an observable are important for the choice of a global pooling layer, because the networks should be able to generalize to other lattice sizes apart from the one they have been trained on. In a regression task, an intensive quantity requires a global average pooling layer, while an extensive quantity calls for a global sum pooling layer. In a classification task, it is not the physical observable itself that is predicted but a decision boundary, so the choice of global pooling layer is more subtle.

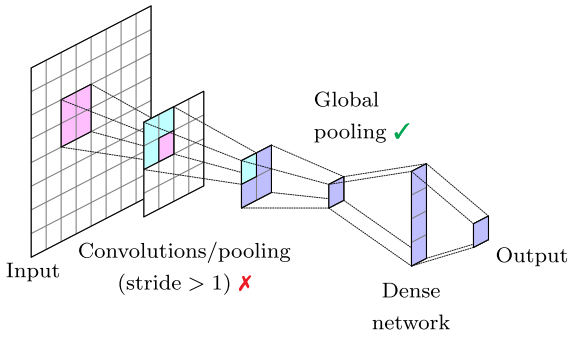
We are interested in the generalization capability to larger lattices, because usually studies on the lattice are intended as an approximation of the real case of an infinite spacetime background. Therefore, a compromise has to be found between lattice size and computational effort such that the simulation produces results satisfactorily close to the physical ones in a reasonable amount of time. The approach adopted throughout this paper is to train models on small lattices and examine how well they generalize to larger lattices.

The three machine learning tasks that are described at the end of Sec. I are tackled with three different types of CNN architecture, which are depicted in Fig. 1:

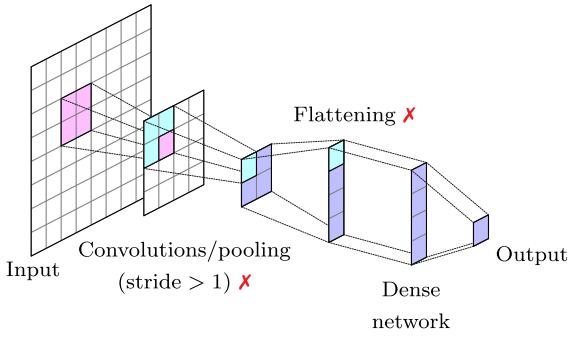
(i) a translationally equivariant architecture (EQ) that uses only layers of stride one and a global pooling layer and is applicable to different lattice sizes,
(ii) a strided architecture (ST) that breaks translational equivariance due to spatial pooling layers with a stride greater than one but is still suited to give predictions on different lattice sizes due to a global pooling layer, and
(iii) a flattening architecture (FL) that represents a "traditional" architecture that breaks translational



(a) Equivariant architecture (EQ)



(b) Strided architecture (ST)



(c) Flattening architecture (FL)

FIG. 1. The three different architecture types used in this study: (a) Equivariant architecture (EQ), (b) Strided architecture (ST), and (c) Flattening architecture (FL). The check mark (✓) or cross (✗) indicate spatial operations in which translational symmetry is respected or violated, respectively. Translational symmetry can be violated by convolutional or pooling layers with a stride greater than one, as in (b) and (c), or by a flattening layer, as in (c). A global pooling layer allows for the application of the same network to different lattice sizes. Each of the layers can have a number of channels (not depicted) without affecting the translational symmetry properties.

equivariance due to spatial pooling layers with a stride greater than one and a flattening layer, which restricts its usage to one particular lattice size.

After the global pooling or flattening step, a dense feed-forward network, which is also known as multilayer perceptron (MLP), can be attached without altering the translational equivariance properties of the network. A discussion about which network layers are translationally equivariant and which ones are not, as well as what exactly breaks said equivariance, can be found in Sec. III.

Dense neural networks can be seen as universal function approximators [34,35], and as such, they do not respect any particular symmetries. Such symmetries can be implemented in (or "hard baked" into, as other authors [10] call it) the network architecture, though, so that the function that is learned is restricted to respect a certain symmetry by design, independent of any training. We expect that such a restriction, as incorporated by EQ architectures, is beneficial and that CNNs of that kind, therefore, outperform CNNs of the other two kinds.

Alternatively, symmetries can be learned, which can be encouraged by augmenting the training data according to the desired symmetry transformation. Thus, we expect data augmentation to improve the performance of networks of ST and FL architectures. Note, however, that, when using data augmentation, it is not guaranteed that the network respects the symmetry even on the training set, let alone on the test set. In addition, data augmentation only establishes a relation between the input layer and the output layer; it does not require the hidden layers in between to respect the symmetry.

From a more theoretical standpoint, a CNN can be seen as a special case of an MLP, where the latter learns to set its weights so that the receptive fields are local (by setting the other weights to zero) and to "share" the appropriate weights (by setting them to the same value). The idea behind the CNN was, though, that this does not have to be learned but can be implemented in the architecture.

Note that if the observable to be studied violates the symmetry that is implemented in the network, it cannot be properly approximated by design. Therefore, it is important to understand the symmetry properties of the task before selecting a particular architecture.

III. SYMMETRY PROPERTIES OF MACHINE LEARNING LAYERS

If a network layer's output is invariant under a symmetry transformation of the input, the outputs of all subsequent layers are invariant under the symmetry transformation of the former layer's input as well. However, invariance of the network's prediction does not require every individual layer to be invariant under this symmetry. The more general concept of equivariance not only is a sufficient requirement, but also allows for more expressive networks. Group equivariant convolutional neural networks (G-CNNs) [8] exploit symmetry transformations that are described by a group $\mathcal{G}$. Conventional CNNs can be seen as a special case of G-CNNs, with the translation group $\mathbb{T}$ as their symmetry group, i.e., $\mathcal{G} = \mathbb{T}$.

The following discussion about equivariance is based on Ref. [8]. The condition for equivariance of a network layer Φ under a group transformation L_g by the element $g \in \mathcal{G}$ is given by

$$\Phi(L_g x) = L'_g \Phi(x). \tag{9}$$

Note that $L_g \neq L'_g$, in general, and that invariance under L_g is the special case $L'_g = 1$.

A. Convolutional layers

On a two-dimensional rectangular lattice, a convolution[1] is defined as

$$[f \star \psi](x) = \sum_{y \in \mathbb{Z}^2} f(y)\psi(y - x) = \sum_{y \in \mathbb{Z}^2} f(x + y)\psi(y), \tag{10}$$

where the feature map f and the kernel (or filter) ψ are real-valued functions:

$$f\colon \mathbb{Z}^2 \to \mathbb{R}, \tag{11}$$

$$\psi\colon \mathbb{Z}^2 \to \mathbb{R}. \tag{12}$$

The kernel ψ is assumed to have finite support $\Psi \subset \mathbb{Z}^2$; i.e., there is only a finite number of points on $\mathbb{Z}^2$ where ψ is nonzero. In principle, this allows us to restrict the sum on the rhs of Eq. (10) to Ψ, but for simplicity we keep the sum running over all of $\mathbb{Z}^2$. For our purposes, the feature map f is understood to be defined on a finite, rectangular proper subset $F \subset \mathbb{Z}^2$ with periodic boundary conditions. We can avoid explicitly dealing with periodic boundaries by assuming that the feature map periodically repeats outside F. In machine learning frameworks such as PyTorch [36], periodic boundary conditions are enforced through the use of circular padding. As a result of periodicity, the output of the convolution Eq. (10) has the same size as the feature map f.

We define a translation of the feature map via

$$[L_t f](x) = f(x - t), \tag{13}$$

where $t \in \mathbb{T}$ is an element of the translation group, which can be identified with an element of $\mathbb{Z}^2$. The convolution is equivariant under translations due to

[1]The cross-correlation in signal processing is often referred to as convolution in the machine learning community. For this paper, we adopt this nomenclature. Additionally, we disregard a possible bias term b to be added to Eq. (10) without loss of generality.

$$[L_t f \star \psi](x) = \sum_{y\in\mathbb{Z}^2} f(y-t)\psi(y-x) = \sum_{y'\in\mathbb{Z}^2} f(y')\psi(y'-(x-t)) = [f\star\psi](x-t) = [L_t[f\star\psi]](x). \quad (14)$$

Equation (10) assumes the convolution to have a stride s of one; i.e., the number of points that the kernel is shifted when the convolution is performed is one. More generally, convolutions with strides $s \geq 1$ can be written as

$$[f\star\psi]_s(x) = \sum_{y\in\mathbb{Z}^2} f(y)\psi(y-sx). \quad (15)$$

This definition reduces to the original convolution if $s = 1$. For $s \geq 2$, the output size of the convolution is smaller than the input size of the feature map f. Strided convolutions with $s \geq 2$ generally break translational equivariance. This can be demonstrated by considering a translation $t \in \mathbb{T}$ with $|t| < s$. For example, we can choose $t = (1, 0)$. Performing this translation on the input feature map f yields

$$[L_t f\star\psi]_s(x) = \sum_{y\in\mathbb{Z}^2} f(y-t)\psi(y-sx) = \sum_{y'\in\mathbb{Z}^2} f(y')\psi(y'-sx+t) = \sum_{y'\in\mathbb{Z}^2} f(y')\psi(y'-s(x-t/s)). \quad (16)$$

In order for the above expression to be equivariant, we would need to be able to rewrite it in terms of a shifted position $x' = x - t/s \in \mathbb{Z}^2$. However, this is not possible, because $t = (1, 0)$ is not divisible by $s \geq 2$. On the other hand, the strided convolutions are equivariant if we consider only the subgroup $\mathbb{T}_s \subset \mathbb{T}$ consisting of translations by multiples of s lattice points. In that case, any element $t \in \mathbb{T}_s$ is divisible by s and, therefore,

$$[L_t f\star\psi]_s(x) = \sum_{y\in\mathbb{Z}^2} f(y-t)\psi(y-sx) = \sum_{y'\in\mathbb{Z}^2} f(y')\psi(y'-s(x-t/s)) = \sum_{y'\in\mathbb{Z}^2} f(y')\psi(y'-sx') = [L_{t'}[f\star\psi]_s](x), \quad (17)$$

where $t' = t/s \in \mathbb{T}$. This means that a convolutional layer with a given stride is equivariant only under translations that are a multiple of that stride. Equivariance under all possible translations is given only for $s = 1$. The generalization to more than one feature map, i.e., multiple channels, is straightforward. Note that a convolution with $s \geq 2$ is equivalent to a convolution with $s = 1$ combined with a subsequent subsampling step.

B. Spatial pooling layers

Spatial pooling layers are usually used to subsample, i.e., $s \geq 2$ in pooling layers. For this discussion, let us split this layer up into a pooling step and a subsampling step. Since average pooling is equivalent to a special case of a convolution, where all weights of ψ are identical and given by $1/|\Psi|$, with $|\Psi|$ denoting the cardinality of Ψ, the average pooling step is equivariant under translations. The subsequent subsampling, however, breaks this equivariance, which again leads to equivariance only under translations that are a multiple of the spatial average pooling layer's stride.

This holds not only for average pooling though, but for spatial pooling, in general: We take again the pooling step by itself or, equivalently, with $s = 1$. It acts on the feature map f by performing the same operation on subsets U_x of F:

$$Pf(x) = \underset{y\in U_x}{P} f(y). \quad (18)$$

These subsets correspond to the kernel of the pooling operation. Its dependence on x depicts the "sliding" of the kernel over the feature map. A spatial pooling step respects Eq. (9), as can be seen by

$$PL_t f(x) = \underset{y\in U_x}{P} f(y-t) = \underset{y'\in U_{x-t}}{P} f(y') = L_t Pf(x). \quad (19)$$

Thus, also in a spatial pooling layer it is the stride that restricts the equivariance of the layer to translations by multiples of said stride.

We want to stress that spatial pooling layers with $s = 1$ respect translational equivariance and can, therefore, be included if one desires an architecture that incorporates such a symmetry, albeit in a different role than usual because it does not subsample.

C. Global pooling

If we wanted to use a traditional CNN architecture on a two-dimensional lattice with periodic boundary conditions, we would have another problem as well: The last convolutional or pooling layer is often flattened and densely connected to the linear layers at the end of the network. Since different positions in one feature map are connected to different weights without a sliding kernel, this is another point where translational equivariance

is broken. A possible solution to this problem is a global pooling layer between the last convolution and the first dense layer. The GAP layer was first introduced in Ref. [6]. There, the authors proposed to create one feature map for each class and to feed the average of each feature map directly to a softmax layer. This approach would respect translational symmetry, although, in general, dense layers could be used between the global pooling and the softmax operation.

D. Equivariant architectures

On the aforementioned two-dimensional lattice with periodic boundary conditions and for similar problems, we propose the following network architecture for classification and regression tasks: The input is fed to a convolutional layer with a stride of one and circular padding so that the output of the convolution has the same size as its input. The kernel size can be odd or even. Translational equivariance is retained by applying consecutive convolutional layers, all with $s = 1$, with nonlinear activation functions in between. Activation functions do not influence the symmetries of an individual layer, since they are applied pointwise. If information from different scales is required, dilated convolutions [37] can be used with a stride of one. Since dilated convolutions are equivalent to convolutions with a larger kernel and the appropriate weights set to zero, they are also equivariant under translations if their stride is one. Spatial pooling layers for subsampling, which use $s > 1$, break translational equivariance, but it is still possible to use them with $s = 1$ between convolutional layers. A way of subsampling that respects translational equivariance is rendered possible by coset pooling [8]. However, since this is a nonlocal operation, we do not expect it to be suitable for the machine learning tasks discussed in this paper, which focus on local quantities and predictions. In the special case of translationally invariant functions, we suggest to utilize a global pooling layer after the last convolution. The output of the global pooling layer is translationally invariant, and, therefore, the rest of the network can be a general MLP without breaking the symmetry.

There is still one important point to be made: Every layer before the GAP respecting translational equivariance is sufficient to guarantee invariance under translations after the GAP, but it is not necessary. If a spatial average pooling layer that breaks translational equivariance and a subsequent convolutional layer are inserted just before the GAP, the output of the GAP can still be invariant under translations, depending on their strides (Theorem 1 in Appendix C). If there is an activation function after the convolutional layer, as is usually the case, the GAP's output is, in general, no longer invariant under translations. The activation function is also necessary for the convolution not to lead to a single multiplicative and additive factor of the GAP, as is shown in Lemma 2 in Appendix C. We thus stick to the aforementioned sufficient conditions for translational equivariance and apply an activation function after the convolutional layer right before the GAP.

IV. REGRESSION: PREDICTING OBSERVABLES ON THE LATTICE

This section revisits a regression task that has previously been performed in Ref. [22]: Given a lattice configuration as input, the network shall predict two physical observables, namely, the particle density n and the lattice averaged squared absolute value of the field $|\phi|^2$. The former is given by

$$n = \frac{1}{N_x N_t} \sum_x k_{x,2}, \tag{20}$$

where the summation of one of the four integer fields $k_{x,2}$ runs over all $N_x N_t$ lattice sites. The latter is given by

$$|\phi|^2 = \frac{1}{N_x N_t} \sum_x \frac{W(f_x + 2)}{W(f_x)}, \tag{21}$$

which contains the highly nonlinear function $W(f_x)$. It is given in Eq. (7) and depends on all four integer fields.

The function $W(f_x)$ also depends on the physical parameters λ, η, and μ, which are set to the same values as in Ref. [22]. Concretely, the values of the coupling constant λ and the mass m will be kept fixed in this task ($\lambda = 1$, $\eta = 4 + m^2 = 4.01$), and the chemical potential μ lies in the interval $\mu \in [0.91, 1.05]$, with steps of $\Delta\mu = 0.005$.

In Ref. [22], the networks have been trained on lattice configurations and observables that have been generated with two values of μ, specifically, the outermost values $\mu = \{0.91, 1.05\}$, but tested on data that have been created on the whole given interval of the chemical potential. This allows for an analysis of the architectures' generalization capability to lattice configurations that correspond to chemical potentials that are not represented in the training set. We will follow this procedure, with the exception that we will use only a single μ to generate training data, namely, the uppermost one $\mu = 1.05$. Since we would test on only smaller values of the chemical potential than the one that has been used for training in our approach, we deviate from Ref. [22] in that we create additional test data that contain higher values of the chemical potential. They lie in the interval $\mu \in [1.1, 1.5]$, with steps of $\Delta\mu = 0.1$. This renders possible an analysis of the architectures' generalization capability to lattice configurations that correspond to values of μ that are greater than the one used to create the training set. We will come back to these test data only at the end of this section. In addition to the generalization ability to different values of the chemical potential, we will investigate the generalization ability to lattice sizes that the models have not been trained on. This highlights a key advantage of

architectures that employ a global pooling layer between their convolutional and their dense layers over architectures that simply flatten the data, because the latter are restricted to a given input size.

A. Architecture choice

The datasets stem from a physical system, whose properties should be taken into account when choosing a network architecture for a model that should learn from said dataset. Let us assume for the following discussion that we have no knowledge of the exact form of Eqs. (20) and (21).

First, the observables are invariant under arbitrary translations of the lattice configuration. This leads to the restriction of preferred architectures that has been proposed at the end of Sec. III: The input is passed to a convolutional layer with a stride $s = 1$ and circular padding that causes its output to have the same size as its input. Such layers are used consecutively, with nonlinear activation functions in between. Optionally, spatial pooling layers with $s = 1$ can be inserted. The output of such convolutional and pooling layers is equivariant under translations of the input. The output of the last convolution is fed to a global pooling layer, which makes it invariant under translations of the input. Then, the data are passed through an MLP with two output nodes, one for each observable.

Second, the observables are derivatives of the logarithm of the partition function on the lattice. The partition function is a product over quantities at each lattice site. The observables can, therefore, be written as a sum over the lattice. Consequently, we want to use a global pooling layer that respects this fact, which excludes global max pooling. Since the observables are intensive quantities and the network shall be able to generalize to different lattice sizes, global average pooling is the natural choice. The MLP at the end does not modify the intensive nature of the prediction.

To check how the above theoretical considerations perform in practice, we want to compare the three types of architecture that are depicted in Fig. 1 from Sec. II. A fair comparison among these network architecture types is quite difficult. One could take a translationally equivariant architecture and break equivariance by inserting at least one spatial pooling layer with $s > 1$. This would keep the number of parameters the same. However, having found a decent EQ architecture, it is not guaranteed that the corresponding ST architecture is a good one compared to other ST architectures and vice versa. Also, keeping the weights constant may not lead to a fair comparison with FL architectures.

Therefore, we define a space of possible architectures for each of the three types separately, which are illustrated in Tables I–III, and use an optimization procedure to find an adequate representative for each architecture type individually.

Table I depicts the search spaces of EQ, Table II of ST, and Table III of FL architectures. The possible parameter values of the first run are inspired by manual trials, which also included different activation functions (ReLU, tanh, PReLU, and LeakyReLU). Its results lead to modifications of the parameter space of the second run and the choice of LeakyReLU for a suitable activation function. Both of them try 50 different combinations of parameters in their optimization procedure on each training set, which will be specified in the next subsection. The extended search explores an enlarged parameter space with 100 trials, also with unique combinations of parameter values, in order to check if a better architecture was missed during the first two runs due to the choice of a too small search space. This search involves only the largest training set. After every

TABLE I. Search spaces for EQ architectures. It lists the possible number of convolutional (conv, $s = 1$) and linear layers (lin), kernel sizes, the number of channels of the convolutional layers, and the number of nodes in the linear layers. Spatial pooling layers with $s = 1$ seem to worsen the predictions and have, therefore, not been included in these search spaces.

	Conv	Lin	Kernel size	Channels or nodes
Run 1	[2, 3]	[0, 1]	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$
Run 2	[2, 4]	1	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$
Extended search	[2, 4]	[0, 3]	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$

TABLE II. Search spaces for ST architectures. It shows the possible number of convolutional ($s = 1$) and linear layers (abbreviated as in Table I), kernel sizes, the number of channels of the convolutional layers, the number of nodes in the linear layers, the number of spatial pooling layers (SPL, $s = 2$), and the spatial pooling mode (SPM).

	Conv	Lin	Kernel size	Channels or nodes	SPL	SPM
Run 1	[2, 4]	[0, 3]	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$	$\{1,2\}$	{avg, max}
Run 2	[2, 4]	[0, 2]	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$	$\{1,2\}$	avg
Extended search	[2, 4]	[0, 3]	$\{(1\times1),(2\times2)\}$	$\{4,8,16,24,32,48,64,80\}$	$\{1,2\}$	{avg, max}

TABLE III. Search spaces for FL architectures. It shows the possible number of convolutional ($s = 1$) and linear layers, kernel sizes, the number of channels of the convolutional layers, the number of nodes in the linear layers, the number of spatial pooling layers ($s = 2$), and the spatial pooling mode. The number of convolutional layers is not chosen directly but follows from the number of 1×1 convolutions that are selected. The asterisk next to "kernel size" signifies that the kernel size of the convolution depends on its position. Two 2×2 convolutions with a respective subsequent spatial pooling layer are mandatory. Additional 1×1 convolutions are possible, namely, before each of the 2×2 convolutions and between each of them and their corresponding subsequent spatial pooling layer. The abbreviations are the same as in Table II.

	Conv	Lin	Kernel size*	Channels or nodes	SPL	SPM
Run 1	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	{avg, max}
Run 2	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	avg
Extended search	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	{avg, max}

convolutional layer and after every linear layer but the last one, a LeakyReLU activation function [38] is applied. Its advantage over the ReLU activation function is the avoidance of so-called dead or dying neurons, which never activate initially or become inactive during the training process.

ST architectures can be thought of as EQ architectures with at least one spatial pooling layer with $s = 2$ in the convolutional part of the network. This is either an average pooling or a max pooling layer, both with a 2×2 kernel. A spatial pooling layer is neither directly applied to the input nor inserted just before the global pooling layer. The position of the spatial pooling layer(s) is part of the search space, but it is restricted by the choice of the number of convolutional layers, as is the number of spatial pooling layers. If, e.g., two convolutional layers are chosen, there can only be one spatial pooling layer at only one specific position, that is, between the convolutional layers.

FL architectures are inspired by how we think one would construct a CNN traditionally for this machine learning problem. At its core are two 2×2 convolutions, followed by a spatial pooling layer with a 2×2 kernel and a stride of 2. Optionally, there can be a 1×1 convolution before each of the 2×2 convolutions and between each of them and the respective following spatial pooling layer, leading to a possible total count of six convolutional layers.

Our optimization procedure of choice has been OPTUNA. The performance metric is the validation loss averaged over three different parameter initializations. This averaging process is applied to counteract the statistical fluctuations introduced by the random initializations of the trainable network parameters. It is important, because OPTUNA changes its search space dynamically, so early search results influence the probability distributions that serve as the basis to select later parameter values. This optimization process is done for different sized training sets individually, since on smaller training sets different architectures might perform better than on larger ones.

After the optimization procedure by OPTUNA, models of the best architectures are retrained ten times from scratch and evaluated on the validation set to verify their performance while further minimizing statistical fluctuations due to the random parameter initializations. Our results show that the same architectures that perform well on small training sets also perform well on larger training sets and that many architectures perform similarly. Thus, we select the best-performing architecture of each type according to the mean validation loss as a representative and compare only them. These best-performing architectures are shown in Table IV. We use $\mathrm{Conv}(K \times K, N_{\mathrm{in}}, N_{\mathrm{out}})$ to denote a two-dimen sional convolution, where K is the kernel size and N_{in} (N_{out}) is the number of input (output) channels. Before every convolutional operation, we use circular padding to enforce periodic boundary conditions. Additionally, we use a stride of one for each convolution. Average pooling layers with kernel size K and stride s are written as $\mathrm{AvgPool}(K \times K, s)$.

TABLE IV. Best architectures for fitting two observables n and $|\phi|^2$ for each type of architecture. This table shows feed-forward networks as found by our OPTUNA searches. The field configuration in the form of $(N_t, N_x, 4)$ tensors is fed into the network at the top. (The batch size is omitted here.) There are two output nodes for the two observables. The last row denotes the number of trainable parameters for each type.

EQ	ST	FL
$\mathrm{Conv}(1 \times 1, 4, 64)$	$\mathrm{Conv}(1 \times 1, 4, 80)$	$\mathrm{Conv}(1 \times 1, 4, 64)$
LeakyReLU	LeakyReLU	LeakyReLU
$\mathrm{Conv}(1 \times 1, 64, 48)$	$\mathrm{Conv}(1 \times 1, 80, 80)$	$\mathrm{Conv}(2 \times 2, 64, 80)$
LeakyReLU	LeakyReLU	LeakyReLU
$\mathrm{Conv}(1 \times 1, 48, 80)$	$\mathrm{Conv}(1 \times 1, 80, 48)$	$\mathrm{AvgPool}(2 \times 2, 2)$
LeakyReLU	LeakyReLU	$\mathrm{Conv}(1 \times 1, 80, 48)$
$\mathrm{Conv}(2 \times 2, 80, 80)$	$\mathrm{AvgPool}(2 \times 2, 2)$	LeakyReLU
LeakyReLU	$\mathrm{Conv}(2 \times 2, 48, 80)$	$\mathrm{Conv}(2 \times 2, 48, 64)$
GlobalAvgPool	LeakyReLU	LeakyReLU
Linear(80, 2)	GlobalAvgPool	$\mathrm{AvgPool}(2 \times 2, 2)$
	Linear(80, 2)	$\mathrm{Conv}(1 \times 1, 64, 24)$
		Flatten
		Linear(360, 24)
		LeakyReLU
		Linear(24, 2)
33202	26370	47394

Dense layers are denoted by Linear$(N_{\text{in}}, N_{\text{out}})$ with N_{in} (N_{out}) input (output) nodes.

B. Training and testing

The training is performed for every model of each of these three architectures and for each training set analogously: Mean squared error (MSE) is selected as a loss function; the total loss is the arithmetic mean of the individual losses, each of which corresponds to one physical observable. It is optimized with the AMSGrad [39] variant of the AdamW optimizer [40] with a vanishing weight decay. Training models on different sized training sets gives us information about the sample efficiency. Limiting the size of training sets is motivated by machine learning tasks where the generation of training samples is costly, for example, in medical applications or in large-scale simulations on supercomputers. The number of training samples in a training set ranges from 100 to 20000, with steps $\Delta = 50$ from 100 to 250, $\Delta = 250$ from 250 to 1000, $\Delta = 500$ from 1000 to 3000, and $\Delta = 1000$ up to 20000 training samples. The corresponding validation sets contain 10% of the amount of the training set's data. The batch size during training was chosen to be 100 for training sets with at least 500 training samples and 50 otherwise. The reason behind this choice is that the algorithm shall be trained with minibatches. To avoid that this approaches batch training for smaller training sets, a smaller batch size is chosen for them. The training lasts between 100 and 1000 epochs; the exact number is determined by early stopping based on validation loss with a patience value of 25. The model is taken at the time it has had the lowest validation loss. An overview of the chosen parameters is given in Table V.

The training takes place on a 60×4 lattice; the first number refers to the temporal dimension and the second to the spatial one. All data in the training set and the validation set have been generated with $\mu = 1.05$.

Both translationally nonequivariant architectures (ST and FL) are trained with and without data augmentation. The training data are augmented by randomly shifting the input data by a number of pixels that is determined by the symmetry properties of the respective architecture. ST architectures contain at most two spatial pooling layers with a 2×2 kernel and a stride of 2, as is shown in Table II. Therefore, they still incorporate translational equivariance under shifts of multiples of 4 (see Sec. III); and the data can be augmented by shifts of [0, 3] in both directions. FL architectures, however, do not incorporate translational equivariance under any shifts of the input; thus, the data have to be augmented by shifts determined by the lattice size, i.e., by [0, 59] in the time direction and by [0, 3] in the space direction.

The testing can be divided into two parts. As a first step, each architecture is evaluated on the same lattice size as it has been trained on, for various values of μ. This checks whether networks of a given architecture are able to generalize to values of μ that are not represented in the training set. Then, the generalization ability to other lattice sizes is investigated. This second step can be done only with architecture types EQ and ST, because FL architectures require a fixed input size.

The test set on the 60×4 lattice contains samples that have been generated with various values of μ, most of which have not been used for the training and validation sets. This test set contains 4000 lattice configurations pertaining to each $\mu \in [0.91, 1.05]$, with steps of $\Delta\mu = 0.005$, where only the last value $\mu = 1.05$ has been used for training and validation. This amounts to 1.16×10^5 testing samples in total.

For testing on different lattice sizes, we generated a test set analogous to the one on the 60×4 lattice on a 50×2, a 100×5, a 125×8, and a 200×10 lattice. For each of these lattice sizes, we created again 1.16×10^5 test samples, 4000 pertaining to each $\mu \in [0.91, 1.05]$, with steps of $\Delta\mu = 0.005$. Note that the winning ST architecture (see Table IV) can be evaluated on the 50×2 lattice, because it contains only one spatial pooling layer with a 2×2 kernel and a stride $s = 2$. Further details on the dataset generation can be found in Appendix B.

C. Results

In this subsection, we will discuss the test results on the 60×4 lattice, which is the lattice size on which the training took place, in detail before analyzing the generalization ability to other lattice sizes of the different network types. Then, we will investigate the silver blaze phenomenon on the larger lattice sizes with our trained models. Finally, we will discuss the results on our second set of test sets, which contains data generated with a chemical potential greater than the one used to create the training set.

TABLE V. Loss, optimizer and early stopping settings for PyTorch.

Loss	Size_avg	Reduce	Reduction		
MSELoss	None	None		"Mean"	
Optimizer	lr	betas	eps	weight_decay	amsgrad
AdamW	0.001	(0.9,0.999)	10^{-8}	0	True
	Monitor	min_delta	Patience	Mode	
EarlyStopping	"val_loss"	0	25	"min"	

1. Results on the same lattice size as training

The loss over the whole test set is a metric for how well the network performs. It is displayed in Fig. 2 for different training sets with a varying number of training samples. Essentially, all of our models are trained until convergence, since we choose a very high number of maximum epochs and employ early stopping based on validation loss. Therefore, the comparison in Fig. 2 shows how the different architectures perform under limited information for smaller training set sizes. The plot at the top shows that the performance of the EQ architecture improves with the size of the training set, as can be expected. The other two architectures do not seem to benefit from increasing the number of samples in the training set, which is quite surprising. Another remarkable result is that data augmentation does not seem to lead to an increase in performance either, as can be seen in the plot in the middle and at the bottom. At first sight, one may draw the conclusion that the ST and the FL architectures do not allow for approximations that are as precise as the one of the EQ architecture. If the model has already converged to an optimal solution, adding more training samples, irrespective of them being newly created or coming from data augmentation, will not improve its performance. However, the blue downward spikes in the loss of the ST model show that some models succeed in finding a good approximation of the observables. Therefore, we draw the conclusion that, although possible, it is more unlikely for the ST and FL models than for the EQ models to learn a good approximation of n and $|\phi|^2$.

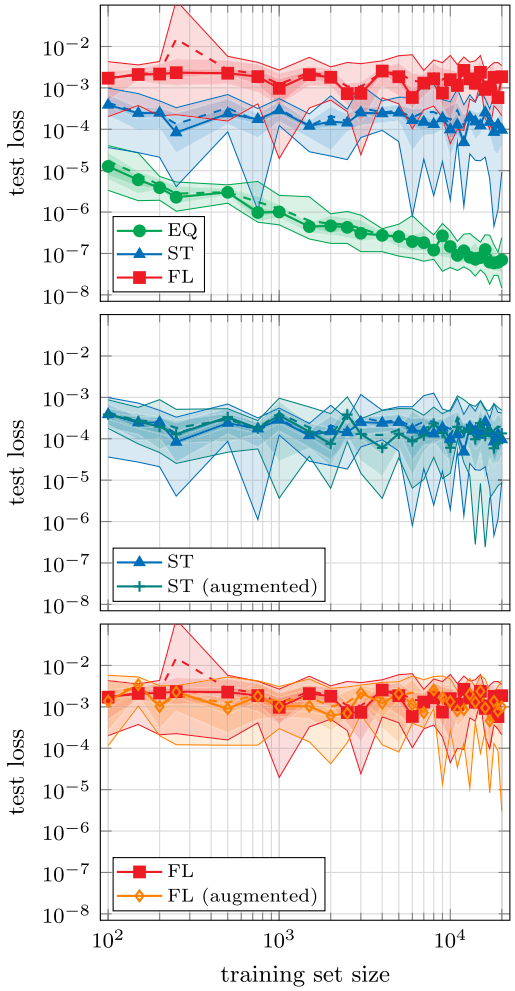


FIG. 2. Test loss on the whole test set on the 60×4 lattice against the size of the training set (number of samples in the training set) on which the respective model has been trained. At the top, the results of the three architecture types (trained without data augmentation) are shown. In the middle and at the bottom, the effect of data augmentation during training of ST and FL models, respectively, is depicted. The plots display the best and worst loss (solid lines), the arithmetic mean of all ten random initializations for training (dashed lines), and the 20% quantiles (shaded regions). The symbols visualize the positions of the measurements; the lines are there to guide the eye.

The predictions of the individual observable's ensemble averages per μ made by the best EQ model, according to the test loss, are displayed in Fig. 3. It shows that the model, although trained only on samples generated with $\mu = 1.05$, can generalize to all other values of the chemical potential in the investigated interval. This seemingly astonishing generalization ability can be understood by recognizing that the network does not need to generalize from one μ to all others but from the training samples to other samples, each consisting of a lattice configuration and two observables. Even though the training set contains only lattice configurations that have been generated with $\mu = 1.05$, the range of possible values for n and $|\phi|^2$ is quite large, and the chosen value of μ in the training set already covers most of the observable values in the test set.

We exemplify this point using ST models that have been trained on 18000 samples: Figure 4 shows the predicted versus the true values of both observables of the best (top)

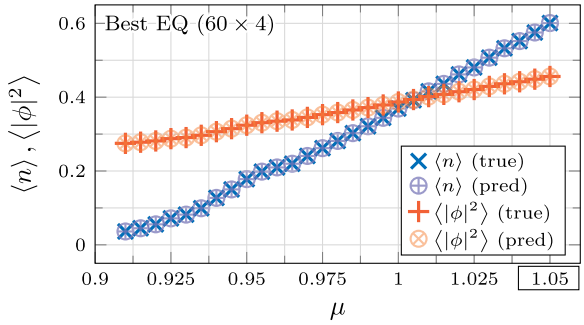


FIG. 3. Predicted and true values for ensemble averages $\langle n \rangle$ and $\langle |\phi|^2 \rangle$ as a function of chemical potential μ on a 60×4 lattice. The predictions in this plot are made by the EQ model with the smallest test loss. The model has been trained on data generated with $\mu = 1.05$ only but shows remarkable generalization capabilities to other values of μ. In this and in subsequent plots, the training point is highlighted by a rectangle.

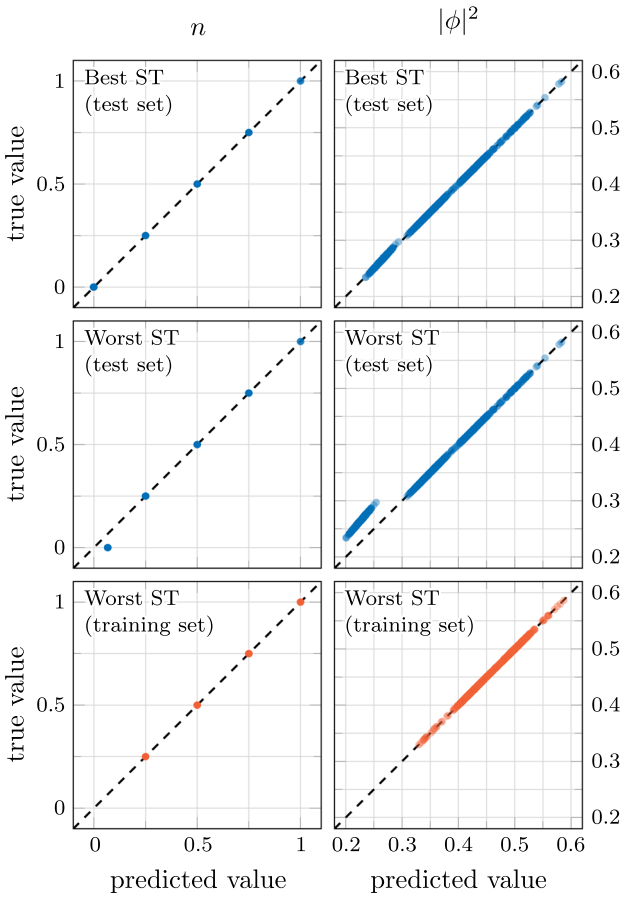


FIG. 4. Predicted versus true observables for the best and the worst ST networks that have been trained on 18000 samples. It shows that the ST architecture's best instance is able to accurately estimate the whole ranges of observable values (top) and that its worst instance is failing to do so for smaller values of n and $|\phi|^2$ (middle). The reason for this is that the training set includes only larger values of the observables (bottom) and that the worst model is not able to generalize beyond that. The top and the middle plot show 1% of the test data; the bottom plot shows 4% of the training data.

and the worst (middle) performing ST model (according to the test loss) evaluated on the test set. The performance of the worst ST model on the training data is shown at the bottom in Fig. 4. Note that in this scatter plot we do not average over the ensemble but show the predictions of the network for each individual example. Both networks are able to predict the larger values of both observables, but the worse one fails to predict the smallest values, since they are missing from the training set. The difference between the better and the worse ST models is the ability to generalize to lattice configurations and ranges of observable values that is has not seen during training. The bad performance overall with some better performing outliers, which is shown in Fig. 2, suggests that ST networks succeed only sometimes with this generalization. FL models show a similar behavior to ST models, but the predictions are less precise throughout. A more detailed discussion of the input value distributions is given in Appendix B.

2. Results on different lattice sizes

One big disadvantage of FL architectures impedes them from predicting on other lattice sizes than the one it was trained on: It requires a fixed input size. For this reason, we can compare only the performance of the EQ and the ST architecture. Since the results for the latter with and without data augmentation are very similar, we will show only the results without data augmentation. Also, we will fix the size of the training set for this comparison to 20000 training samples.

Figure 5 displays the overall test loss (top) and the individual losses of the observables (middle and bottom). Even though the ST architecture keeps its worse performance from the 60×4 lattice, the generalization ability to the different lattice sizes is comparable for the EQ and the ST architecture, with the exception of the 100×5 lattice for the latter. This kink in the blue curve shows up in the prediction of both observables, whereas this particular lattice size does not seem to be extraordinary to the EQ architecture. The problem is the odd number in the lattice dimension. This behavior can be explained by a closer inspection of the ST

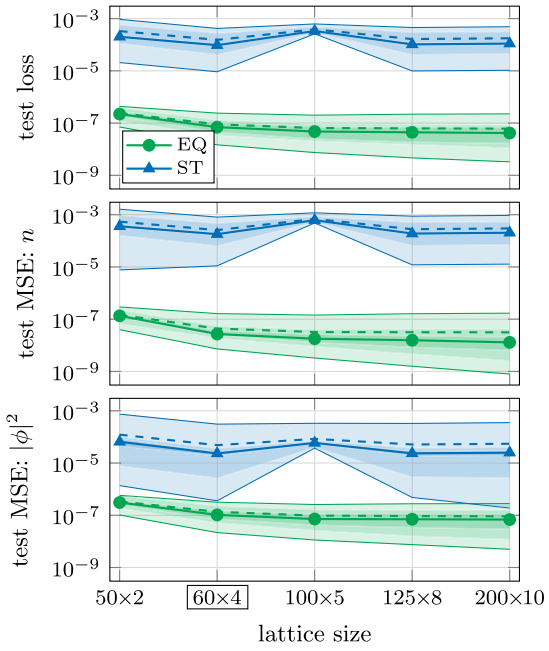


FIG. 5. Overall test loss (top) and its two parts (middle and bottom) that come from each observable, on various lattice sizes. The training has taken place on the 60×4 lattice. Both architectures generalize well to lattice sizes different from the one they were trained on, but the ST architecture (blue) performs visibly worse on the 100×5 lattice. The reason for this is the spatial pooling layer within the architecture, which drops 20% of the data, leading to a less accurate prediction for both observables.

architecture: The first three convolutions leave the input size unchanged, because they employ circular padding. Then, these 100×5 data are passed to a spatial pooling layer with a 2×2 kernel and a stride of 2. This layer disregards 20% of the data and outputs data with a shape of 50×2. Consequently, the ST networks cannot use all of the data to come to a prediction, which is, therefore, less precise. This is far less severe on the 125×8 lattice, because there the spatial pooling layer disregards only $1/125$ of the data, which is not enough to be visible in Fig. 5. A more detailed analysis of said kink in the blue curve can be found in Appendix D.

3. Silver blaze phase transition

The silver blaze [41] phenomenon refers to a second-order phase transition at vanishing temperature T, where thermodynamical observables are independent of the chemical potential μ below a critical value μ_c [32]. This means that the observables $\langle n \rangle$ and $\langle |\phi|^2 \rangle$ are constant for $\mu < \mu_c$, whereas they start rising if the chemical potential surpasses its critical value. The particle density $\langle n \rangle$ is an order parameter of the silver blaze phase transition. As a result of the finiteness of our lattices, the temperature is nonzero ($T \propto 1/N_t$, where N_t is the number of lattice sites in the time direction), and, thus, the transition is not necessarily sharp. Because of the networks being trained to approximate the particle density and the lattice averaged squared absolute value of the field, this phase transition should also be visible in their predictions.

Figure 6 visualizes predictions of the EQ architecture model that has been trained on 20000 training samples and reached the lowest validation loss. More precisely, it shows the mean prediction of each observable for each individual μ, as well as the true mean value, on the 100×5 (top), the 125×8 (middle), and the 200×10 (bottom) lattice. The largest lattices show both phases, whereas the smaller lattices show no phase transition in the range of μ that we analyzed. This is because μ_c decreases for increasing temperature.

The silver blaze phase transition is also predicted correctly by the ST models that accurately generalize to the smaller values of the observables, e.g., by the model that is shown at the top in Fig. 4, but not all ST models generalize well.

4. Extrapolation to larger chemical potentials

After inspecting the remarkable results that the EQ architecture and some models of the ST architecture achieved on the interval $\mu \in [0.91, 1.05]$, the question remains as to how the different architectures perform on data corresponding to chemical potentials greater than the one of the training set.[2] To answer it, we evaluate the already trained networks on additional test sets, without retraining them. We have created one test set for each lattice size under consideration. Each of them contains 4000 lattice configurations corresponding to each $\mu \in [1.1, 1.5]$, with steps of $\Delta\mu = 0.1$. This amounts to 2×10^4 test samples per lattice size.

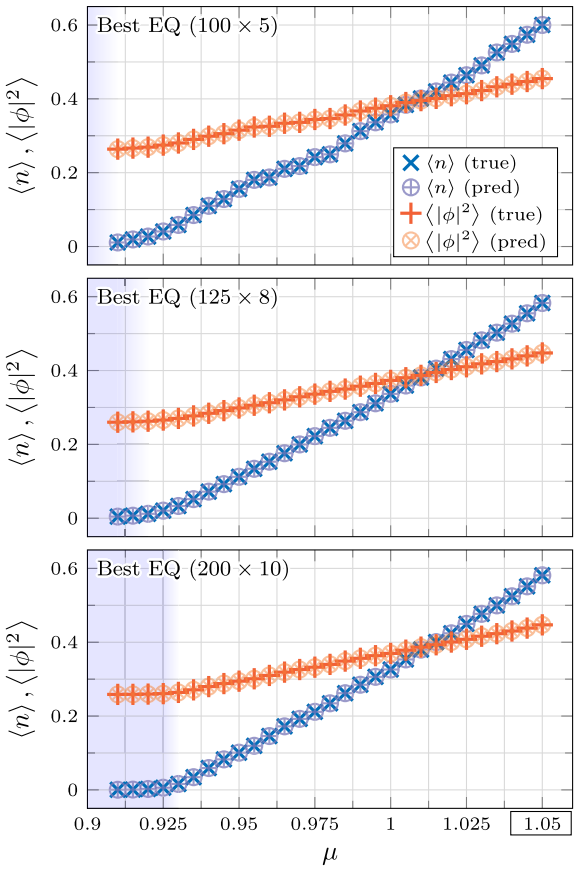


FIG. 6. Predicted and true mean values of each observable for each individual μ on the larger lattices. The predictions come from the EQ model that has the lowest validation loss from all EQ models that have been trained on 20000 training samples. The training has been performed at $\mu = 1.05$. The kinks in the curves allow for an estimate of the silver blaze phase transition, which is indicated by the color gradient from the shaded region to the white background.

The predicted versus the true values of both observables on the 60×4 lattice are shown in Fig. 7. The individual rows correspond to the respective best model of each architecture, according to the validation loss. Although the extrapolation to higher $|\phi|^2$ seems to be more difficult than to higher n, the predictions of the EQ architecture's best model remain close to the identity line, and they are visually better than the predictions of the other two architectures' best models, the FL model performing the worst. This leads to a visible deviation in the ensemble averages of the observables only for $\mu = 1.5$ and is comparable on all lattice sizes under consideration, with the exception of the FL architecture, which allows only for predictions on the 60×4 lattice without adapting the architecture and retraining. Note that "best" refers to the validation loss and that there are models

[2]This was done thanks to a suggestion by the referee.

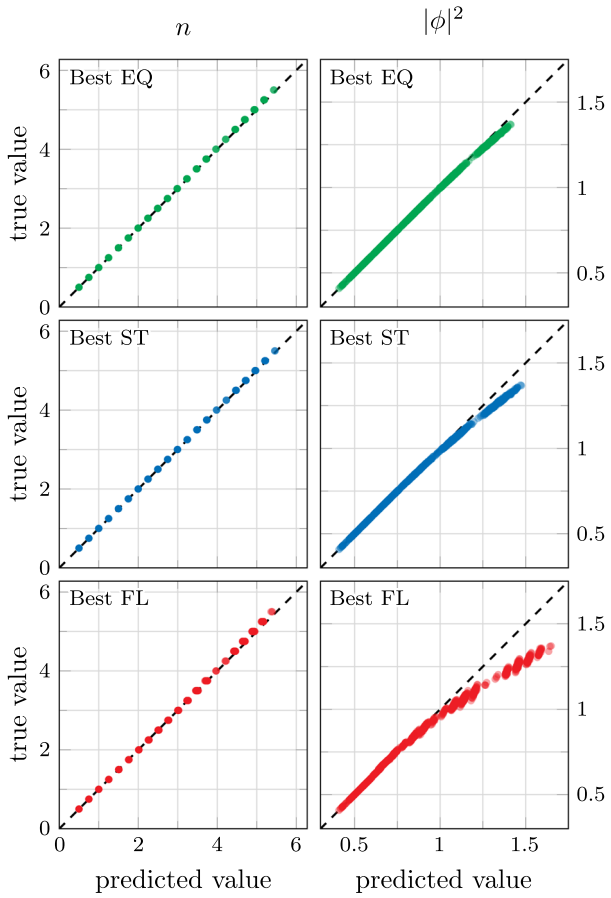


FIG. 7. Predicted versus true observables for the best (according to the validation loss) model of each architecture evaluated on the test set generated from $\mu \in [1.1, 1.5]$ on the 60×4 lattice. Each model is able to predict higher values of n than given during training, but the generalization of $|\phi|^2$ exhibits a clear difference between the generalization capabilities of the models. All these plots show 6.25% of the test data.

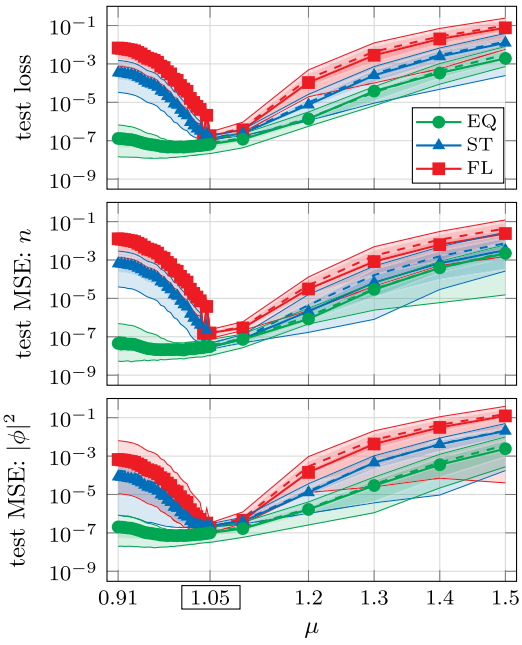


FIG. 8. Total test loss and its parts corresponding to the individual observables n and $|\phi|^2$ over the chemical potential on the 60×4 lattice. It displays the ensemble of models that have been trained on 20000 training samples corresponding to each architecture. The large difference in the quality of the predictions for $\mu \leq 1.05$ is also visible in Fig. 2. For $\mu > 1.05$, the performance is more similar, although, for $\mu = 1.5$, the mean value of the total test loss of the ST models still differs from the mean values of the other architecture's models by roughly one order of magnitude.

of each architecture that extrapolate better than the respective ones depicted in Fig. 7. However, since we are analyzing the generalization capabilities of the networks, we are restricted to metrics that take into account only the training and the validation data, and we chose the validation loss.

Figure 8 shows the total and individual test losses over μ on the 60×4 lattice. While the large difference between the different architectures in performance on chemical potentials smaller than $\mu = 1.05$ is quite substantial, the performance on larger values of μ differs by less. At $\mu = 1.5$, for example, the mean and median losses of the EQ architecture are lower than their respective counterparts belonging to the other architectures, but there the ST architecture's best model leads to the lowest. An analogous comparison between the EQ and the ST architecture on other lattice sizes leads to similar results, with the exception of the 100×5 lattice, on which the latter fails. Note that Fig. 7 depicts the model with the lowest validation loss pertaining to each individual architecture. For the EQ architecture, it is a model of the ensemble that has been trained on 20000 training samples, whereas for the ST and the FL architectures, it is a model that has been trained on 18000 training samples. Figure 8, however, shows the ensemble of models trained on 20000 training examples for each individual architecture.

5. Results summary

In summary, the best translationally equivariant architecture performs better than the respective best model of the other two types on the lattice size that they have been trained on. Only some ST networks are able to generalize beyond values of observables that they have encountered during training, while EQ networks show no such problem. The FL architecture shows similar behavior to the ST architecture, but its predictions are less precise overall. Models of FL architectures cannot be applied to different lattice sizes. The EQ and the ST architectures are both capable of generalizing to different lattice sizes, although the latter retains the higher average test loss from the 60×4 lattice due to the bad generalization to observable values that were not in the training set. Furthermore, ST architectures are not suited to

make predictions on every arbitrary lattice size. Each lattice dimension has to regard the behavior of the spatial pooling layers in the network in order to use all the data for the prediction. EQ architectures have the advantage to impose no such restriction. Even though all the models have been trained only on $\mu = 1.05$ on the 60×4 lattice, many of them are able to predict the silver blaze phase transition on a different lattice size, where $\mu_c \ll 1.05$. The EQ architectures do this especially well. We found that data augmentation does not help in the training of ST and FL architectures, which is why we refrain from using it in the next two tasks.

Lastly, we want to make a comparison to the results of Ref. [22] where the same regression task was performed. Our best model needs much fewer trainable parameters than the one in Ref. [22], i.e., approximately 3×10^4 compared to over 10^7 as extracted from their network architecture. We also found well-performing models that contain by an order of magnitude fewer parameters than our best one. Furthermore, their network architecture would fall in our FL category, which means that it can be employed on only one specific lattice size.

V. CLASSIFICATION: DETECTING FLUX VIOLATIONS

In the previous section, we have found that rather simple CNN models can approximate the functions n and $|\phi|^2$ sufficiently well. In fact, the function n can be exactly represented by a linear, equivariant model with a single 1×1 convolution. Similarly, while $|\phi|^2$ does not admit an exact representation in terms of 1×1 convolutions, it is easy to see that the lattice averaged quantity $\sum_x |\phi_x|^2/(N_t N_x)$ can be written as a sum over a function that receives dominant contributions from $k_{x,\mu}$ and $l_{x,\mu}$ at the same lattice site x.

In order to study models that require larger kernel sizes, we need to shift our focus to quantities that cannot be computed by taking into account only field values at a single lattice site. One example for such a quantity is the local flux violation

$$\mathcal{F}_x \equiv \sum_{\nu=1}^{D} (k_{x,\nu} - k_{x-\hat{\imath},\nu}) \in \mathbb{Z}. \tag{22}$$

Evaluated at some lattice site x, it specifically requires information from nearest neighbors surrounding x.

We therefore propose to solve the following classification task: An arbitrary field configuration $X = \{k_{x,\mu}, l_{x,\nu}\}$ is mapped to the label $y(X)$:

$$y(X) = \begin{cases} 0 & \mathcal{F}_x = 0, \ \forall \ x, \\ 1 & \text{else.} \end{cases} \tag{23}$$

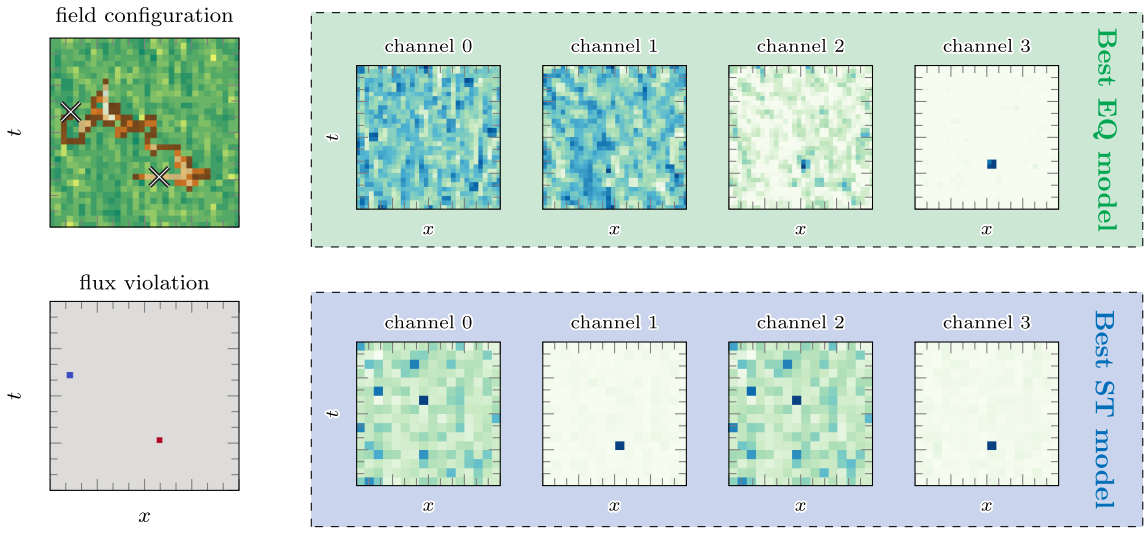
Since the worm algorithm generates only physical field configurations which by design satisfy the flux constraint $\mathcal{F}_x = 0, \ \forall \ x$, we adapt it to generate configurations including open worms. The field configurations generated this way exhibit flux violations at each end of the open worm (see Fig. 9). While we will be using such open worm configurations only for the purpose of classification and regression tasks, they are typically utilized in the calculation of n-point functions of ϕ [42,43].

For this task (and the following counting task in Sec. VI), we have generated field configurations on square lattices of various sizes given by $(N_t \times N_x) \in \{8 \times 8, 16 \times 16, 32 \times 32, 64 \times 64\}$. The value of the coupling constant is fixed to $\lambda = 1$, the mass m takes values given by $\eta = 4 + m^2 \in \{4.01, 4.04, 4.25\}$, and possible values of the chemical potential μ are given by $\mu \in \{1, 1.25, 1.5\}$. Training is performed only on the smallest lattice size (8×8) and two specific choices for the pair (η, μ): $(\eta_1, \mu_1) = (4.25, 1)$ and $(\eta_2, \mu_2) = (4.01, 1.5)$. We use a fixed number of training examples, $N_{\text{train}} = 4000$, distributed equally between the two classes: On half of the generated field configurations, we generate an open worm on top of a flux-constraining configuration. Other combinations of parameters and lattice sizes are used only during testing. Further details regarding the datasets can be found in Appendix B.

A. Architecture search, training, and testing

We aim to make a comparison between the three different architecture types that have been presented in Fig. 1. As discussed previously, both EQ and ST architectures can be applied to field configurations of varying lattice size, while FL architectures are compatible only with a fixed lattice size. As we are dealing with a binary classification problem, a sigmoid activation function is applied to the output of our models.

To facilitate a fair comparison among architecture types, we use OPTUNA to perform a search for well-performing architectures using validation loss (binary cross entropy loss) as the metric to optimize for. In all three cases, we allow for up to $N_{\text{conv,max}} = 3$ convolutional layers with circular padding and a maximum kernel size of $K = 3$ and $N_{\text{ch}} \in \{4, 8, 16, 32\}$ possible channels. Every convolution is followed by applying a LeakyReLU activation function. In addition, after every convolution except the last, we allow for the insertion of a pooling layer (either average or max pooling) with stride $s = 1$ in the case of EQ networks and $s = 2$ in the case of ST and FL networks. For nonequivariant architectures, we require at least one pooling layer with $s = 2$ to break translational equivariance. Following this convolutional part of the network, we either apply a global max pooling layer (EQ and ST) or flatten the remaining lattice structure (FL). Although other global pooling layers are possible (e.g., average pooling or sum pooling), global max pooling seems to be the most fitting choice when the task is to detect pointlike defects in the field configuration. We note that, as an additional search parameter, we allow for explicitly setting bias terms to zero in every convolutional layer. The resulting feature map is



(a) Example field configuration

(b) Feature maps of convolutional network in best EQ and ST models

FIG. 9. Visualization of an open worm field configuration and of the best models' predictions. (a) An example field configuration including an open worm (highlighted in brown) and the resulting flux violation given by Eq. (22). The pointlike violations occur at the two open ends of the worm (shown as crosses). (b) Feature maps of the convolutional part of the best EQ (top, green) and ST (bottom, blue) model showing the first four channels (of 32 and 16, respectively). Because of overparameterization, only some of the channels detect the violation (e.g., channels 2 and 3 for EQ and 1 and 3 for ST), while other channels (e.g., 0 and 1 for EQ and 0 and 2 for ST) do not produce easily interpretable output.

then fed to a dense network with up to $N_{\text{dense,max}} = 2$ layers with $N_{\text{nodes}} \in \{4, 8, 16, 32\}$ nodes. Every linear layer is followed by the application of LeakyReLU. A final linear layer is used to map the activation values to a single output node, which is followed by a sigmoid activation function. As before, we use binary search parameters for setting bias terms to zero in each linear layer.

For each architecture type, we perform two OPTUNA search runs with 400 trials each. Each model candidate (i.e., a set of hyperparameters) is trained five times with randomly initialized weights to reduce random fluctuations from the stochastic optimization algorithm. Among the two searches, the best-performing architecture (according to validation loss) is chosen and retrained 50 times to build an ensemble of models for each architecture type.

Training proceeds similar to the regression task in Sec. IV. We use the AMSGrad variant of the AdamW optimizer without weight decay, a learning rate of $\lambda_{lr} = 10^{-3}$, a batch size of 100, and 200 epochs. We employ early stopping based on validation loss with a patience value of 50. The validation set consists of 2000 examples from the same distribution as the training set.

The best architectures found during the OPTUNA search for each type are shown in Table VI.

B. Results

Our main results are presented in Figs. 10 and 11. Figure 10 shows a comparison of all three architecture types evaluated on 8×8 lattices as a function of μ. Both EQ and ST exhibit very good classification accuracy and test loss, while our ensemble of FL models contains a few outliers which increase the average test loss. Figure 11, which shows an average (loss and accuracy) of all available test datasets, demonstrates that both EQ and ST architectures generalize well on larger lattices. FL architectures are not included, since they can be used for

TABLE VI. Best architectures for detecting flux violations. This table shows feed-forward architectures as found by our OPTUNA searches. Input in the form of $(N_t, N_x, 4)$ tensors is fed into the network at the top. The output of each network is a classification probability. The last row denotes the number of trainable parameters for each type. We use an asterisk (*) to denote layers where the bias is explicitly set to zero.

EQ	ST	FL
Conv$(2 \times 2, 4, 32)$	Conv$^*(2 \times 2, 4, 16)$	Conv$^*(3 \times 3, 4, 8)$
LeakyReLU	LeakyReLU	LeakyReLU
Conv$(1 \times 1, 32, 32)$	MaxPool$(2 \times 2, 2)$	MaxPool$(2 \times 2, 2)$
LeakyReLU	Conv$(1 \times 1, 16, 16)$	Conv$(2 \times 2, 8, 32)$
GlobalMaxPool	LeakyReLU	LeakyReLU
Linear$(32, 32)$	Conv$(1 \times 1, 16, 8)$	AvgPool$(2 \times 2, 2)$
LeakyReLU	LeakyReLU	Conv$(2 \times 2, 32, 32)$
Linear$^*(32, 1)$	GlobalMaxPool	LeakyReLU
Sigmoid	Linear$^*(8, 32)$	Flatten
	Linear$(32, 1)$	Linear$^*(128, 1)$
	Sigmoid	Sigmoid
2657	953	5600

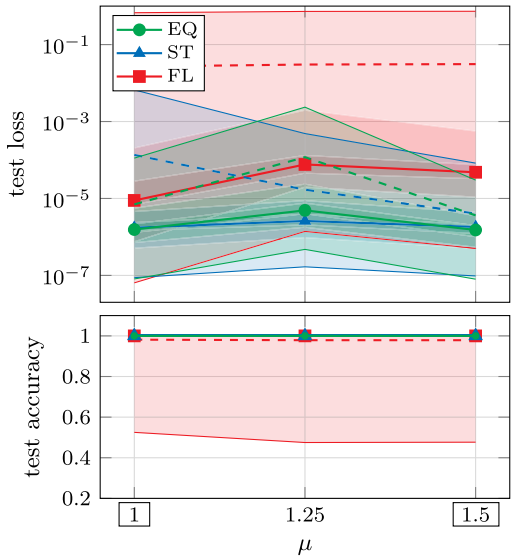


FIG. 10. Top: test loss for best equivariant (EQ, green), nonequivariant strided (ST, blue), and nonequivariant flattening (FL, red) classification architectures as a function of the chemical potential μ on 8×8 lattices. Training was performed on data with $\mu = 1$ and 1.5 only. Bottom: test accuracy as a function of μ. The colored bands show the ensemble uncertainty from all 50 randomly initialized models with the thick line indicating the median loss (accuracy) and the dashed line showing the mean loss (accuracy). Both EQ and ST architectures outperform the FL architecture.

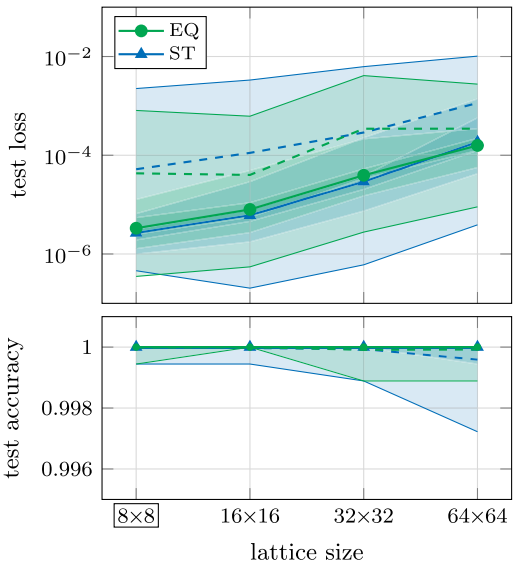


FIG. 11. Top: test loss for best equivariant (EQ, green) and nonequivariant strided (ST, blue) classification architectures as a function of lattice size. Bottom: test accuracy as a function of lattice size. The networks have been trained on the 8×8 lattice only. We observe that both types of architecture lead to good generalization across lattice sizes with slightly less variation in the performance of the EQ architecture.

only one specific lattice size (in this case, 8×8). It is evident that FL models perform worse on average compared to EQ and ST networks, but, in contrast to the previous regression task, the nonequivariant architecture without flattening (ST) exhibits similar performance to the equivariant architecture (EQ). The loss of spatial information due to pooling operations with stride $s > 1$ does not seem to affect the ability of ST models to correctly classify flux violations.

In light of the results in Figs. 10 and 11, the question arises how EQ and ST models are able to make predictions with such high accuracy and if the computation that is performed by the networks can be easily understood and interpreted. To answer this, we "dissect" fully trained EQ and ST models by examining the feature maps that are generated by the convolutional part of the network; i.e., we modify models by removing the global pooling operation and the dense network. Examples of these feature maps are shown in Fig. 9(b). We see that some of the channels of the output of the convolutional part of the network highlight flux violations in the vicinity of one of the open ends of the worm [see Fig. 9(a)]. At first, it seems surprising that only one of the two open ends is detected. However, one has to keep in mind that the models were not directly trained on the local flux violation as in Eq. (22) but instead were only given global information about whether or not a field configuration contains a violation. Detecting a single defect is sufficient to make the correct prediction.

It is further noteworthy that, compared to typical deep learning models, the models found in our architecture search are rather small, with ∼2700 parameters in the case of EQ models and ∼1000 parameters for our best ST architecture on this task.

VI. REGRESSION: COUNTING FLUX VIOLATIONS

A natural extension of last section is the study of lattice configurations with more than one open worm, meaning a regression task where the inputs are lattice configurations that are labeled by the number of open worms N_{worms} that they contain. The addition of an open worm implies the emergence of a flux violation at its head and tail, meaning that the quantity defined in Eq. (22) respects $\mathcal{F}_x = \pm 1$ at an open worm end point. As discussed in more detail in Appendix B, we explicitly forbid end points of different worms to lie on top of each other; therefore, a configuration with N_{worms} open worms is characterized by $2N_{\text{worms}}$ points where $\mathcal{F}_x = \pm 1$. With this clarification, the task we are going to tackle in this section can be formally expressed as the approximation of the function

$$y(X) = \frac{1}{2}\sum_x |\mathcal{F}_x|, \tag{24}$$

where X is a lattice configuration $\{k_\mu, l_\nu\}$. We note that this task resembles a simplified version of other counting problems, such as crowd counting [44].

The physical parameters are the ones mentioned in the previous section, with the addition of a number of open worms ranging from 0 to 10, yielding a total of $36 \times 11 = 396$ combinations of parameters. While the test set includes data coming from all these combinations, the training set consists of data created at only a small subset of such combinations to inspect the generalization capabilities of the architecture under consideration. We use a training set with $N_{\text{train}} = 20000$ samples distributed equally between two different numbers of open worms $N_{\text{worms}} \in \{0, 5\}$ and physical parameters $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$. The validation set contains $N_{\text{val}} = 2000$ samples. For more details regarding the datasets, see Appendix B.

A. Architecture search, training, and testing

A preliminary phase is carried out in order to explore trends with different hyperparameter choices. We also empirically confirm the relationship between the prediction of an extensive quantity and the necessity of a global sum after the convolutional part of the neural network, as discussed in Sec. II. The information gathered in this initial stage is then used to determine the architecture search space for OPTUNA. As in the two previous tasks, this is done for the three architecture types shown in Fig. 1. The search spaces are designed to be as similar as possible to eliminate favorable conditions for any of the three architecture types.

The EQ architecture search space is characterized by $N_{\text{conv}} \in \{2, 3, 4\}$ convolutional layers with a kernel size $K \in \{1, 2, 3\}$, followed by a global sum pooling layer which leads to a dense network, composed of $N_{\text{dense}} \in \{0, 1, 2\}$ layers. The ST architecture search space is structured in the same way with the additional insertion of $N_{\text{pool}} \in \{1, 2\}$ spatial pooling layers with stride $s = 2$. Since training is conducted on 8×8 lattices, three such pooling layers would reduce the lattice to only one site and render global sum pooling ineffective, which is why we limit the choice of N_{pool}. The FL architecture search space features two mandatory convolutions with a 2×2 or a 3×3 kernel, each followed by a spatial pooling layer. A 1×1 convolution can be inserted before and after each mandatory convolution, leading to a total number of convolutions $N'_{\text{conv}} \in \{2, 3, 4, 5, 6\}$. This part is followed by the flattening layer and a dense network consisting of $N'_{\text{dense}} \in \{1, 2, 3\}$ layers, where the maximum number of layers is increased with respect to the other two architecture types to compensate for the possible absence of 1×1 convolutions. All three types share the following features: Circular padding is used in every convolution; the channels in the convolutions and the nodes in the dense layers are selected from the set $N_{\text{ch/nodes}} \in \{4, 8, 16, 32\}$; a LeakyReLU activation function is used after every convolution and every linear layer not leading to the output; the bias in both the convolutions and the linear layers is turned off. We also mention that an independent search is run also for EQ architectures with the optional inclusion of spatial pooling layers with stride $s = 1$, in the same fashion described for ST models. However, none of the EQ models found in this run are better than the EQ models found in the previous search.

As in the previous section, two metrics are employed for performance analysis: the MSE loss and the accuracy, for which predictions are rounded to the closest integer. The quantity monitored during the optimization phase is validation loss. Since the hyperparameter search spaces are large, two OPTUNA runs are executed to reduce the risk of overlooking promising regions. For each hyperparameter selection, three models are trained, in order to attenuate initialization influences, for 200 epochs with no early stopping. The other hyperparameters are defined prior to the optimization: We adopt a batch size of 16, a learning rate $\lambda_{lr} = 10^{-3}$, and the AMSGrad variant of the AdamW optimizer with zero weight decay.

Out of 100 different architectures from the two OPTUNA searches, the best three for each type are selected according to the validation loss averaged over their three initializations. These architectures become the starting point of the next step: training the most promising architectures from scratch.

We keep all the same hyperparameters, except for the number of epochs which is increased to 500, and the same training and validation sets. For a fair comparison, 20 instances of the same architectures are trained to mitigate the influence of random initializations, and for each of them the best model is saved. We sort the architectures according to the average over the 20 models of the validation loss. Table VII portrays the details of the feed-forward networks.

B. Results

Since FL models cannot be evaluated on input sizes different from the ones they have been trained on, we make two kinds of comparison between architectures: One involves all three types tested only on 8×8 lattices, and the other focuses on EQ and ST tested on all lattice sizes available. The first analysis is featured in Fig. 12 and the second in Fig. 13, where the dashed lines indicate the mean values and the markers represent the medians.

A common takeaway of these plots is that for this task equivariance proves to be an important property to incorporate into the network. Interestingly, Fig. 12 suggests that ST and even more so FL have difficulties in recognizing certain numbers of open worms, with the lowest performance at $N_{\text{worms}} = 1$, which is compatible with the fact that the training set consists only of $N_{\text{worms}} \in \{0, 5\}$.

We observe that the podium ordering depends on the metric chosen; for example, in Table VIII, the mean and the median of the validation loss lead to different

TABLE VII. Best architectures for counting flux violations. This table lists the feed-forward architectures resulting from the OPTUNA searches sorted by their average validation loss over 20 instances trained from scratch. Four channels of size $N_t \times N_x$ are the input tensors passed at the top of each network, which yields a scalar output representing the predicted number of open worms. The last row shows the number of trainable parameters for each architecture.

1st EQ	2nd EQ	3rd EQ
Conv$(1\times1,4,32)$	Conv$(2\times2,4,8)$	Conv$(1\times1,4,4)$
LeakyReLU	LeakyReLU	LeakyReLU
Conv$(2\times2,32,8)$	Conv$(2\times2,8,8)$	Conv$(2\times2,4,8)$
LeakyReLU	LeakyReLU	LeakyReLU
Conv$(2\times2,8,16)$	Conv$(1\times1,8,4)$	Conv$(2\times2,8,4)$
LeakyReLU	LeakyReLU	LeakyReLU
Conv$(1\times1,16,8)$	Conv$(1\times1,4,8)$	Conv$(3\times3,4,1)$
LeakyReLU	LeakyReLU	LeakyReLU
GlobalSumPool	GlobalSumPool	GlobalSumPool
Linear(8, 1)	Linear(8, 1)	
1800	456	308

1st ST	2nd ST	3rd ST
Conv$(2\times2,4,16)$	Conv$(2\times2,4,4)$	Conv$(2\times2,4,4)$
LeakyReLU	LeakyReLU	LeakyReLU
Conv$(1\times1,16,32)$	MaxPool$(2\times2,2)$	AvgPool$(2\times2,2)$
LeakyReLU	Conv$(2\times2,4,4)$	Conv$(3\times3,4,16)$
Conv$(1\times1,32,32)$	LeakyReLU	LeakyReLU
LeakyReLU	GlobalSumPool	GlobalSumPool
AvgPool$(2\times2,2)$	Linear(4, 1)	Linear(16, 32)
Conv$(1\times1,32,8)$		LeakyReLU
LeakyReLU		Linear(32, 1)
GlobalSumPool		
Linear(8, 32)		
LeakyReLU		
Linear(32, 1)		
2336	132	1184

1st FL	2nd FL	3rd FL
Conv$(2\times2,4,4)$	Conv$(2\times2,4,8)$	Conv$(2\times2,4,32)$
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool$(2\times2,2)$	AvgPool$(2\times2,2)$	AvgPool$(2\times2,2)$
Conv$(3\times3,4,8)$	Conv$(3\times3,8,4)$	Conv$(3\times3,32,4)$
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool$(2\times2,2)$	AvgPool$(2\times2,2)$	AvgPool$(2\times2,2)$
Flattening	Flattening	Flattening
Linear(8, 4)	Linear(4, 4)	Linear(4, 32)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(4, 32)	Linear(4, 32)	Linear(32, 16)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(32, 1)	Linear(32, 1)	Linear(16, 1)
640	640	2704

winners for all architecture types. As the training and validation sets are characterized by the same physical parameters, which represent a very small subset of the whole set of parameters used for testing, metrics on the validation set may not be indicative of the generalization capabilities of the network, so we investigate the same metrics on the test set in the two manners described at the beginning of this subsection.

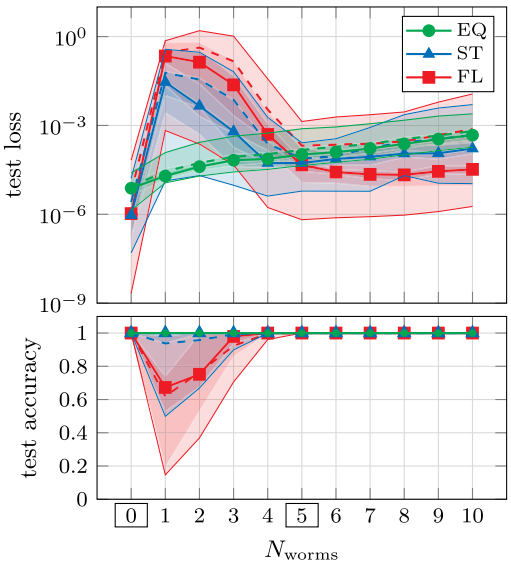


FIG. 12. Test loss (top) and test accuracy (bottom) of the best architectures according to the mean of the validation loss tested on all 8×8 lattices as functions of the number of open worms. Training and validation are carried out at $N_{\text{worms}}=0$ and $N_{\text{worms}}=5$, while test results are shown for $N_{\text{worms}}\in[0,10]$.

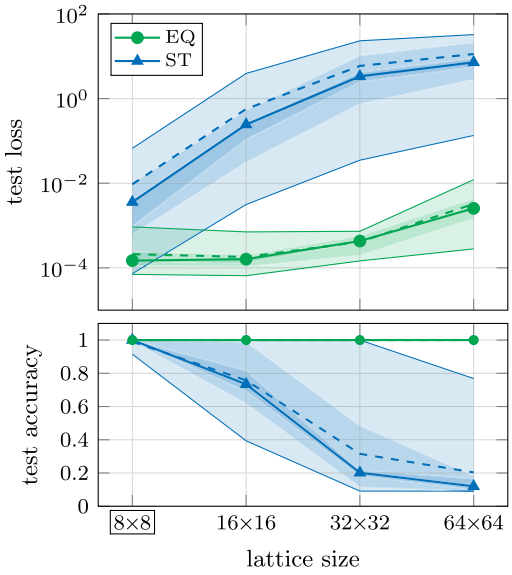


FIG. 13. Test loss (top) and test accuracy (bottom) of the best architectures according to the mean of the validation loss tested as functions of the lattice size. Training and validation are carried out on the smallest lattice (8×8), while testing is performed on all lattice sizes.

An in-depth analysis is shown in Figs. 14 and 15, depicting the relationship between the test and validation

TABLE VIII. Metrics of the best architectures for counting flux violations. Highlighted in bold are the results of the best architectures for each type according to the corresponding metric.

	Validation loss on 8×8		Test loss on 8×8		Test loss up to 64×64	
	Mean	Median	Mean	Median	Mean	Median
1st EQ	$\mathbf{4.676 \times 10^{-5}}$	4.137×10^{-5}	$\mathbf{2.108 \times 10^{-4}}$	1.483×10^{-4}	$\mathbf{1.008 \times 10^{-3}}$	8.308×10^{-4}
2nd EQ	1.042×10^{-4}	$\mathbf{2.440 \times 10^{-5}}$	3.525×10^{-4}	$\mathbf{8.783 \times 10^{-5}}$	1.807×10^{-3}	$\mathbf{7.936 \times 10^{-4}}$
3rd EQ	8.992×10^{-3}	3.072×10^{-4}	2.105×10^{-2}	9.163×10^{-4}	1.925	4.031×10^{-2}
1st ST	$\mathbf{2.331 \times 10^{-5}}$	2.173×10^{-5}	9.438×10^{-3}	3.576×10^{-3}	4.446	3.026
2nd ST	8.479×10^{-5}	4.372×10^{-5}	$\mathbf{2.545 \times 10^{-4}}$	$\mathbf{9.340 \times 10^{-5}}$	$\mathbf{3.738 \times 10^{-3}}$	$\mathbf{1.171 \times 10^{-3}}$
3rd ST	2.869×10^{-4}	$\mathbf{2.171 \times 10^{-5}}$	1.676×10^{-2}	1.381×10^{-3}	2.943	9.580×10^{-1}
1st FL	$\mathbf{2.602 \times 10^{-5}}$	1.787×10^{-5}	7.837×10^{-2}	3.817×10^{-2}		
2nd FL	4.004×10^{-5}	1.117×10^{-5}	$\mathbf{5.300 \times 10^{-2}}$	$\mathbf{1.285 \times 10^{-3}}$		
3rd FL	5.805×10^{-5}	$\mathbf{1.031 \times 10^{-5}}$	6.382×10^{-2}	3.556×10^{-2}		

loss for all 20 models of each architecture. If an architecture is prone to generalization issues, its instances are scattered mostly vertically, which manifestly happens to most of the ST and FL architectures. EQ models are instead distributed closer to the black line, where test and validation loss are equal. These results suggest that for EQ architectures low validation loss correlates with low test loss, and, therefore, they tend to reliably generalize.

Remarkably, there is also a nonequivariant architecture featuring this behavior, specifically, the 2nd ST, whose best two instances even outperform the best EQ models by almost an order of magnitude both in the validation and in the test loss. This is an illustration that the validation procedure does not guarantee generalization if the validation set is restricted to a small set of physical parameters. Indeed, the test loss on 8×8 lattices in the central column in Table VIII already contains a generalization in terms of physical parameters, which implies that it would be sufficient to include at least some of those configurations in the validation set in order to select the best generalizing architecture on different lattice sizes.

We observe two distinctive properties of this outstanding ST architecture, that are possible contributing factors to its success: It is the only one containing a spatial max pooling layer, and it is characterized by a very small number of parameters, the smallest of all the examined architectures, as can be seen in Table VII. Max pooling can be beneficial for the detection of local defects, as pointed out in the

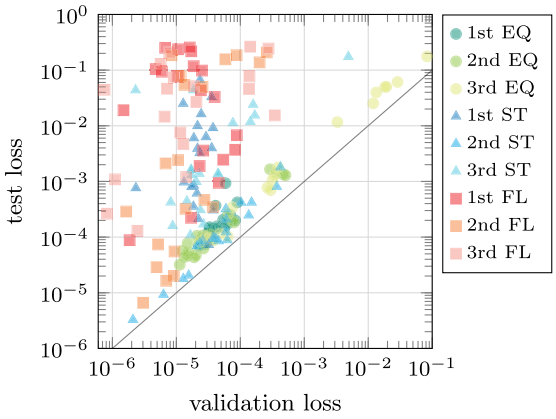


FIG. 14. Test loss on 8×8 lattices versus validation loss of every instance for each architecture. This scatter plot shows 20 models obtained during retraining for the three winning architectures of each type (EQ, ST, and FL). The diagonal black line indicates where validation loss equals test loss. Networks have been trained and validated for $N_{\text{worms}} \in \{0, 5\}$ and $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$ on an 8×8 lattice. Generalization (test loss) is checked with zero to ten open worms, $\mu \in \{1.0, 1.25, 1.5\}$, $\eta \in \{4.01, 4.04, 4.25\}$, and a fixed lattice size of 8×8. The closer a particular point lies to the black line, the better it generalizes. This appears to be generally the case for EQ architectures (green circles).

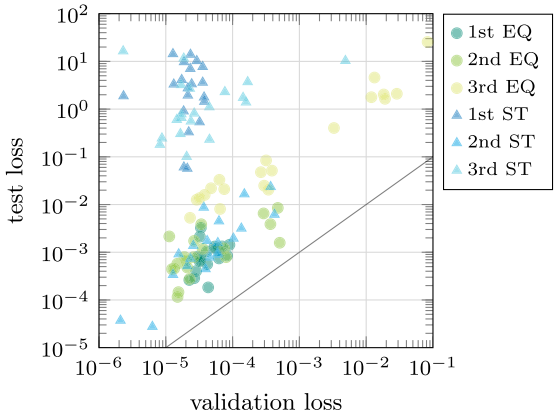


FIG. 15. Scatter plot of test loss on all lattice sizes versus validation loss of every instance for EQ and ST architectures. Similar to Fig. 14, we demonstrate the generalization capabilities of our models to different lattice sizes from 8×8 up to 64×64 and different physical parameters, while being trained on only 8×8. In particular, EQ models (green circles) are closer to the black line where test loss and validation loss agree.

previous task, while the relative simplicity of this counting problem calls for simpler network structures. Indeed, OPTUNA favors overall small architectures for this task, too, with a number of parameters between ∼100 and ∼3000 for the ones studied in detail.

VII. CONCLUSIONS AND OUTLOOK

In this work, we studied the effect of imposing global translational invariance on convolutional neural network architectures. We did so by comparing three different architecture types that are commonly used and which differ with regards to their equivariance and generalization properties. Network architectures that use only convolutions or pooling operations of stride one and a global pooling layer before a subsequent dense network preserve translational equivariance. Such networks are also able to generalize to different input sizes if the global pooling operation is compatible with the intensive or extensive property of the output quantity. Network architectures that contain pooling operations with a stride greater than one generally break translational equivariance. Using a flattening operation instead of global pooling further impairs translational equivariance of the network and restricts its usage to one particular input shape, preventing a straightforward generalization to other lattice sizes without retraining. This latter architecture type has been particularly popular in image classification tasks and has subsequently also been used in physics applications.

We chose three different tasks related to characterizing complex scalar field configurations on a two-dimensional lattice with periodic boundary conditions that are given in the flux representation. This representation contains integer-valued field configurations which have to obey a flux conservation law. Valid configurations are generated using the worm algorithm. The first task we performed was a regression task to predict the particle density n and the field average of $|\phi|^2$, given just the plain field configurations in flux quantities. The predicted observables also depend on various physical parameters, including the chemical potential μ, that are set during the generation of the configurations. We found that it is sufficient to train at only one value of μ in order to be able to generalize to other values of μ, in particular, also to extrapolate beyond the silver blaze phase transition. While this result seems surprising at first, it can be explained by the fact that different input configurations at fixed training chemical potential μ_{train} already cover a wide range of possible input values that are shared between physical parameters, i.e., other values of μ. Comparing the three architecture types by selecting their best-performing representatives from a network architecture search using OPTUNA, we generally find that equivariant architectures perform best at this task. They excel, in particular, when increasing the size of the training set. We also explored whether data augmentation on the input side can compensate for missing equivariance, but this turned out to have a barely noticeable effect on the result. Furthermore, we investigated the generalization properties to smaller and larger lattice sizes, which is possible only for architectures that contain a global pooling layer. Again, across all lattice sizes, the equivariant architecture wins. Strided architectures can generalize only to lattice sizes that are multiples of the stride combinations used in the network, whereas flattening architectures are not able to generalize at all to other lattice sizes. These architectures would have to be retrained for each input size separately.

The next two tasks were related to detecting and counting flux violations from open worm configurations. Such configurations can appear in the calculation of n-point functions. They are particularly interesting, as the result cannot be approximated by a purely local function, which would involve only 1×1 convolutions but requires at least a 2×2 convolution. In the case of detecting flux violations, flattening models perform worst, while equivariant and strided architectures with global pooling layers are both able to predict the result with comparably high accuracy. Inspecting the feature maps of the trained models, we found that these models learn to detect only one end of the open worms, but this is sufficient to solve this task. For the third task to count the number of flux violations, we trained only on configurations containing either zero or five open worms and tested on configurations that contained any number of worms from zero to ten. In this setup, again the equivariant architecture wins compared to strided or flattening architectures. Interestingly, the networks have most problems to differentiate between zero and one worms, while a larger number of worms poses fewer problems. Another interesting observation is that the selection procedure of the network architecture search can lead to different optimal choices with very different generalization properties. Because of our particular choice of validation and test data, the validation loss alone is not sufficient to select the architecture that can generalize best to other physical parameters or network sizes. The optimal models we found were much smaller regarding the number of weight parameters than models used in comparable studies in the literature.

Based on our findings, we can clearly recommend using global pooling layers in future machine learning tasks that involve systems with global translational invariance. Global pooling layers allow one to easily generalize results to different lattice sizes in regression and classification tasks. Whether the advantages of using pooling layers with a stride greater than one outweigh the possible disadvantages of breaking translational equivariance depends on the system being studied. An interesting aspect that warrants further study is the question of why some architectures seem to generalize better than others and whether there is a way to identify or characterize such architectures already before testing on an extended test set. Moreover, physical parameters may not be the best quantities for assessing

generalization capabilities, but one should rather study the distribution of input and output values of the network. While in this work we concentrated on translational symmetry, it would be interesting to extend this study in the future to further symmetries on the lattice using, for example, G-CNNs. One could also examine coset pooling at intermediate layers that respect translational invariance. Finally, based on our findings, it seems worthwhile to investigate and study possible translationally equivariant versions of current architectures that explicitly break translational invariance, even though the underlying theory would respect this symmetry.

The code and datasets used in this work are published in a separate repository [45].

ACKNOWLEDGMENTS

We thank Kai Zhou for correspondence. D. I. M. thanks Jimmy Aronsson for valuable discussions regarding group equivariant neural networks. This work has been supported by the Austrian Science Fund FWF No. P32446-N27, No. P28352, and doctoral program No. W1252-N27. The Titan V GPU used for this research was donated by the NVIDIA Corporation.

APPENDIX A: THE COMPLEX SCALAR FIELD

The action of a $1+1$-dimensional complex scalar field ϕ in the continuum with quartic interaction, a nonzero chemical potential μ, and no external sources can be written as

$$S=\int \mathrm{d}x_0\mathrm{d}x_1(|D_0\phi|^2-|\partial_1\phi|^2-m^2|\phi|^2-\lambda|\phi|^4), \tag{A1}$$

with $D_0=\partial_0-i\mu$, the mass m, and the coupling constant λ. The invariance property of the action under translations in time and space gives rise to the conservation of the energy momentum tensor. After a Wick rotation

$$x_0\to ix_2, \qquad x_2\in\mathbb{R}, \tag{A2}$$

we obtain the imaginary time version of the action in Eq. (A1), namely,

$$S_E=\int \mathrm{d}x_1\mathrm{d}x_2(|\partial_1\phi|^2+|D_2\phi|^2+m^2|\phi|^2+\lambda|\phi|^4), \tag{A3}$$

with $D_2=\partial_2+\mu$ and the imaginary time x_2.

The Euclidean action, which is given by Eq. (A3), can be discretized, which makes it possible to analyze the complex scalar field on the lattice. The result reads (see, e.g., [32])

$$S_{\mathrm{lat}}=\sum_x\Bigg(\eta|\phi_x|^2+\lambda|\phi_x|^4-\sum_{\nu=1}^{2}(e^{\mu\delta_{\nu,2}}\phi_x^*\phi_{x+\hat{\nu}}+e^{-\mu\delta_{\nu,2}}\phi_x^*\phi_{x-\hat{\nu}})\Bigg), \tag{A4}$$

where $\eta=2D+m^2=4+m^2$ and $\delta_{\nu,2}$ is the Kronecker delta. The first sum is over all lattice sites x, and the second one is over the two directions: space and imaginary time. The position $x+\hat{\nu}$ is reached by moving one unit vector $\hat{\nu}$ from x in the ν direction. Naturally, periodic boundary conditions are employed. In Eq. (A4), we have explicitly set the lattice spacing to unity. This implies that all dimensionful quantities such as m and μ are understood to be given in appropriate units of the lattice spacing. We limit the extension of the system to L in the spatial direction and to $1/T$ in the temporal one, where T denotes the temperature.

For nonzero chemical potential μ, the action in Eq. (A4) becomes complex. This is problematic, because in this case the term $e^{-S_{\mathrm{lat}}}$ cannot be interpreted as a probability distribution, and, therefore, it is not possible to use standard Monte Carlo sampling to determine the partition function

$$Z=\int \mathcal{D}\phi e^{-S_{\mathrm{lat}}} \tag{A5}$$

and its derivatives. To circumvent this so-called complex action problem, which is also known as the sign problem, one can work in a dual formulation, known as flux representation. The derivation of the partition function in the flux representation can be found in Ref. [32]. The result reads

$$Z=\sum_{\{k,l\}}\Bigg(\prod_{x,\nu}\frac{1}{(|k_{x,\nu}|+l_{x,\nu})!l_{x,\nu}!}\Bigg)\Bigg(\prod_x e^{\mu k_{x,2}}W(f_x)\Bigg)\times\Bigg(\prod_x\delta\Bigg(\sum_\nu(k_{x,\nu}-k_{x-\hat{\nu},\nu})\Bigg)\Bigg), \tag{A6}$$

with

$$\sum_{\{k,l\}}=\prod_{x,\nu}\sum_{k_{x,\nu}=-\infty}^{\infty}\sum_{l_{x,\nu}=0}^{\infty}=\sum_{k_{1,1}=-\infty}^{\infty}\sum_{l_{1,1}=0}^{\infty}\sum_{k_{1,2}=-\infty}^{\infty}\cdots\sum_{l_{N,2}=0}^{\infty}, \tag{A7}$$

where the N lattice sites have been labeled with numbers $x\in\{1,2,\ldots,N\}$. The degrees of freedom are the four integer fields $k_{x,\nu}$ and $l_{x,\nu}$, where $\nu=1,2$. The former must obey the flux conservation law

$$\sum_{\nu}(k_{x,\nu} - k_{x-\hat{\nu},\nu}) = 0 \tag{A8}$$

at all lattice sites x for the Kronecker delta not to vanish; the latter are non-negative. The function $W(f_x)$ is given by

$$W(f_x) = \int_0^\infty \mathrm{d}x x^{f_x+1} \mathrm{e}^{-\eta x^2 - \lambda x^4}, \tag{A9}$$

and its integer valued argument reads

$$f_x = \sum_{\nu}[|k_{x,\nu}| + |k_{x-\hat{\nu},\nu}| + 2(l_{x,\nu} + l_{x-\hat{\nu},\nu})]. \tag{A10}$$

Observables can be derived from the partition function and written in terms of the dual variables $k_{x,\nu}$ and $l_{x,\nu}$. In this paper, two quantities are of special interest, namely, the particle number density n and the lattice averaged squared absolute value of the field $|\phi|^2$. Their ensemble averages $\langle \cdots \rangle$ are given by

$$\langle n \rangle = \frac{T}{V}\frac{\partial \ln Z}{\partial \mu} = \frac{1}{N_x N_t}\left\langle \sum_x k_{x,2} \right\rangle, \tag{A11}$$

$$\langle |\phi|^2 \rangle = -\frac{T}{V}\frac{\partial \ln Z}{\partial \eta} = \frac{1}{N_x N_t}\left\langle \sum_x \frac{W(f_x+2)}{W(f_x)} \right\rangle, \tag{A12}$$

where N_x (N_t) is the number of lattice sizes in the spatial (temporal) direction.

For our machine learning tasks, we associate each individual configuration $\{k_{x,\mu}, l_{x,\mu}\}$ with particular values of n and $|\phi|^2$ in Eqs. (20) and (21), even though the dual formulation does not allow for a direct mapping between field configurations ϕ_x and link configurations $\{k_{x,\mu}, l_{x,\mu}\}$.

APPENDIX B: DATASETS

In this Appendix, we discuss the Monte Carlo procedure we use to generate the datasets for our machine learning tasks.

The flux representation, which is given by Eq. (A6), is characterized by the positive field l and the field k constrained by Eq. (A8). Since they are different in nature, a suitable algorithm is composed of two distinct parts, each of which takes care of the modifications of the respective field. The link variables l are updated using a standard Monte Carlo algorithm, where the Metropolis acceptance probabilities are ratios of Boltzmann weights of the dual action. The links k are updated by means of the worm algorithm, originally proposed in Ref. [33], where the acceptance probabilities follow the prescriptions given in Ref. [32]. Using these algorithms, we generate all datasets in this work.

The initial configuration is set to zero at every lattice site for both k and l. Before reaching equilibrium, the system undergoes a thermalization phase, which we discard. Since autocorrelation in the dataset can affect the learning process, we monitor it and set an appropriate number of waiting sweeps between each measurement.

1. Regression: Predicting observables on the lattice

The dataset contains lattice configurations and corresponding n and $|\phi|^2$ values, the first ones being the input for the CNN and the latter being the quantities to predict. We create data with the following set of physical parameters: $\eta = 4.01$, $\lambda = 1$, and $\mu \in \{0.91, \ldots, 1.5\}$, where values in the range [0.91, 1.05] are separated by $\Delta\mu = 0.005$, while $\Delta\mu = 0.1$ in the range [1.1, 1.5]. We choose five different lattice sizes: 50×2, 60×4, 100×5, 125×8, and 200×10, where the first number is $N_t = 1/T$ and the second one $N_x = L$. Different N_t means different temperature T, which influences the properties of the phase transition, as shown in Fig. 4. The total amount of training data is $N_{\text{train}} = 20000$, generated at $\mu = 1.05$ on the 60×4 lattice, and the whole validation set consists of $N_{\text{val}} = 2000$ at the same μ and lattice size. We define two distinct test sets, both containing 4000 data points per each μ at each lattice size. The first test set (test set A) is characterized by values of $\mu \in \{0.91, \ldots, 1.05\}$, which correspond to the ones used in Ref. [22], in order that a direct comparison with the results found there is possible. The second test set (test set B) is designed to examine the extrapolation abilities of the neural networks to chemical potentials higher than the one they have been trained on, specifically, in the range $\mu \in \{1.1, \ldots, 1.5\}$. The total amount of test data is $N_{\text{test}} = 4000 \times 5 \times (29 + 5) = 680000$. We discard the first 1000 sweeps to disregard thermalization and then save a configuration and the respective observables every five sweeps. For some combinations of chemical potential and lattice size, we observe a high autocorrelation. In these cases, the number of sweeps is increased to 50, which sufficiently reduces the autocorrelation.

We now closely inspect the distribution of the two fundamental quantities needed for the computation of the observables n and $|\phi|^2$, namely, $k_{x,t}$ and f_x. This is meant to give an additional insight into the dataset properties and the generalization capabilities of the architectures. We remind that $k_{x,t}$ can take any integer value, while f_x is either 0 or a positive even number. In the following discussion, we omit the lattice index x and use k_t, k_x, and f instead.

In Figs. 16 and 17, the first histogram corresponds to the distribution of the training set, while the second and third show the distributions of test sets A and B, respectively. The domains covered by the training set and test set A are approximately the same, meaning that the generalization we require does not involve an extrapolation. Given this, it is easy to see why models that perform well during training and validation are able to generalize to different physical parameters. Despite having the same domain, there is an evident discrepancy in the distributions of the training set

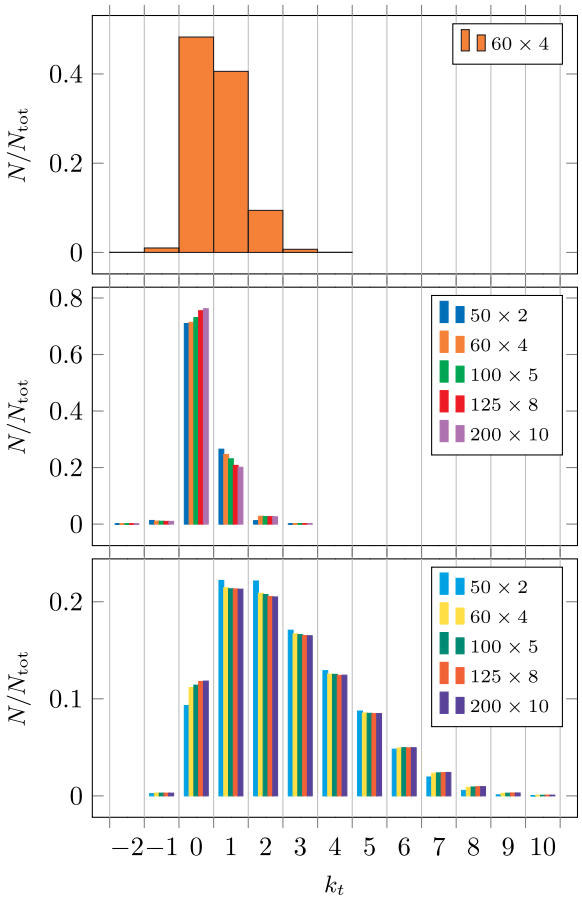


FIG. 16. Distributions of the link field k_t. These histograms feature the distributions of k_t in the training set (top), test set A (middle), and test set B (bottom). The test sets maintain a similar distribution along different lattice sizes. Even though training and test set A cover the same domains, their distributions are different, which is the origin of the generalization issues of some architectures. The distribution of test set B also reaches higher values of k_t, which can make a generalization to data in test set B even more difficult than to data in test set A. Bars corresponding to weights smaller than 10^{-4} in each plot are not shown.

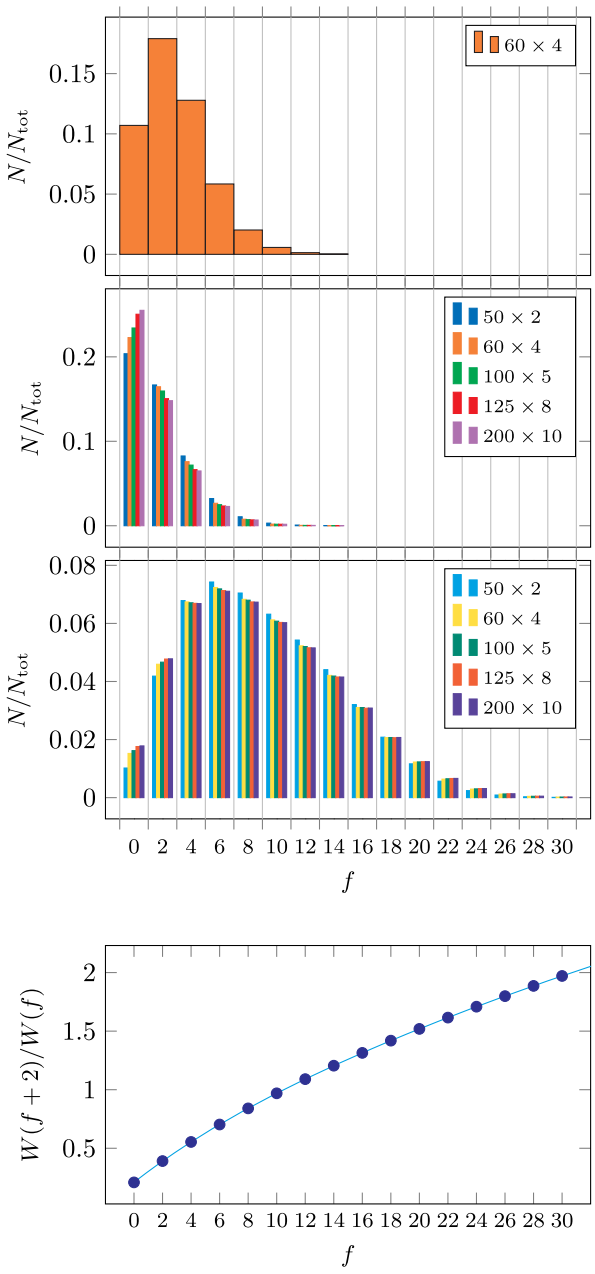


FIG. 17. Distributions of f and ratio of $W(f)$. The histograms show the distributions of f in the training set (top), test set A (middle), and test set B (bottom). The last plot portrays $W(f+2)/W(f)$ evaluated with the same physical parameters used throughout the task, $\eta = 4.01$ and $\lambda = 1$. The markers represent even integer values of f, which enter the computation of $|\phi|^2$. In every histogram, we do not report weights below 10^{-4}.

and test set A, which is caused by the different values of μ that are used to generate the lattice configurations. This could be a possible source of generalization issues affecting some architectures. Note that, while the distributions appear to be somewhat similar, there may still be additional differences in the correlations of these quantities, which could further impair generalization capabilities of networks. The domain that is covered by test set B, however, is larger than the domain of the training set, arguably making the generalization to these data more demanding.

While the relationship between $\langle n \rangle$ and k_t is linear, as shown in Eq. (A11), $\langle |\phi|^2 \rangle$ is highly nonlinear in f, as indicated by Eqs. (A12) and (A9). One might find it surprising that a CNN is able to learn such a complicated function and even generalize to other physical parameters. Alongside the observations on domain and distributions, we have to consider the ratio $W(f+2)/W(f)$ that enters in Eq. (A12). As shown in Fig. 17, it can be effectively approximated by a linear function in the range where most of the distribution of the training set and test set A is concentrated. This explains why even simple models can easily learn to predict $|\phi|^2$ on these data. The larger values of f that are represented in test set B, however, lead to

larger values of said ratio. At these values, the linear approximation that might be a good approximation on the training set and test set A worsens, and so do the predictions of $|\phi|^2$.

2. Classification: Detecting flux violations

The algorithm presented in Ref. [32] is designed to generate only closed worm configurations, which respect the flux conservation and allow one to compute the observables n and $|\phi|^2$. For the classification and regression tasks in Secs. V and VI, we want to create field configurations where the flux conservation (A8) is violated. In order to do this, we modify the algorithm of the previous subsection in the following manner: After equilibrium is reached via the original l and k alternate update, we start a new worm and save the configuration with one open worm. As the worm moves on the lattice, we replace the stored configuration with probability $1/L$, where L is the current worm length, until the worm closes. One can easily check that this corresponds to selecting one of the open worm configurations with equal probability.

The dataset consists of closed worm configurations, labeled as class 0, and open worm configurations, labeled as class 1, each originating from two independent runs of the algorithm. Both classes are characterized by the same physical parameters, namely, $\eta \in \{4.01, 4.04, 4.25\}$, $\lambda = 1$, $\mu \in \{1, 1.25, 1.5\}$, and $N_t = N_x \in \{8, 16, 32, 64\}$. The training set is generated on a particular subset, specifically, the two combinations $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$ on the smallest lattice size, i.e., 8×8, with a total number of $N_{\text{train}} = 4000$ samples equally distributed among each class and parameter combination. The validation set has the same structure, the only difference being the number of samples of $N_{\text{val}} = 400$. The test set contains 100 instances per each class and parameter combination, summing up to $100 \times 2 \times 36 = 7200$ samples. The number of skipped configurations to avoid picking samples while the thermalization process is still ongoing is chosen as 2000. We use 100 waiting sweeps between each measurement. The dataset created for this task and the next one share very similar characteristics. We address the analysis of only the third task in the following subsection, implying that the considerations we make there are also valid in this context.

3. Regression: Counting flux violations

The algorithm designed in the previous task is extended to account for multiple worms. After the first configuration with an open worm is saved as described in the last section, it becomes the starting configuration for the next worm to be drawn. We explicitly prohibit that a worm can cross heads and tails of previous worms, i.e., lattice sites where the flux is violated. By doing this, we ensure the absence of mathematical ambiguity in the definition of the Metropolis acceptance probability. As a consequence, three values for the flux are possible: 0, +1, and −1. The procedure is repeated until the required number of open worms is reached and the configuration is saved. Then the last configuration without open worms is restored, the established waiting sweeps are performed, and another set of open worms is drawn.

The same set of physical parameters of the previous task is used with the addition of the number of worms $N_{\text{worms}} \in \{0, 1, \ldots, 10\}$. The subset of parameters for the training set is chosen as the combinations $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$ and $N_{\text{worms}} \in \{0, 5\}$, again on the smallest lattice size. The total amount of data used is $N_{\text{train}} = 20000$ when training only on 0 and five worms. The validation set consists as usual of a number of configurations such that $N_{\text{val}} = N_{\text{train}}/10$ of the number of training samples. The test set contains again 100 samples per parameter combination, leading to a total of $100 \times 11 \times 36 = 39600$ instances. Initially skipped configurations and waiting sweeps are the same as in the previous task.

The two quantities necessary for the computation of the flux are the two integer fields k_t and k_x, as suggested by Eq. (A8). Their distributions are depicted, respectively, in Figs. 18 and 19. We can draw the same conclusions as in the first task concerning the similarity of the domains and the difference in the distributions between training and test sets. We add that the choice of (η, μ) for the training set is made specifically to include the lower and higher values of k_t, in such a way that the domain covered is the same in the training and in the test set. This explains the two peaks in the k_t training distribution in the top histogram in Fig. 18. Such behavior does not emerge in the case of k_x because, unlike k_t, it is not coupled with the chemical potential.

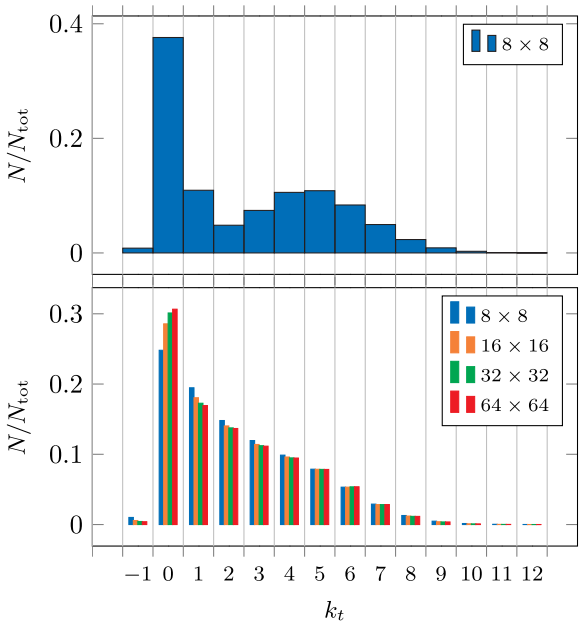


FIG. 18. Distributions of the link field k_t. These two histograms feature the distributions of k_t in the training set (top) and in the test set (bottom).

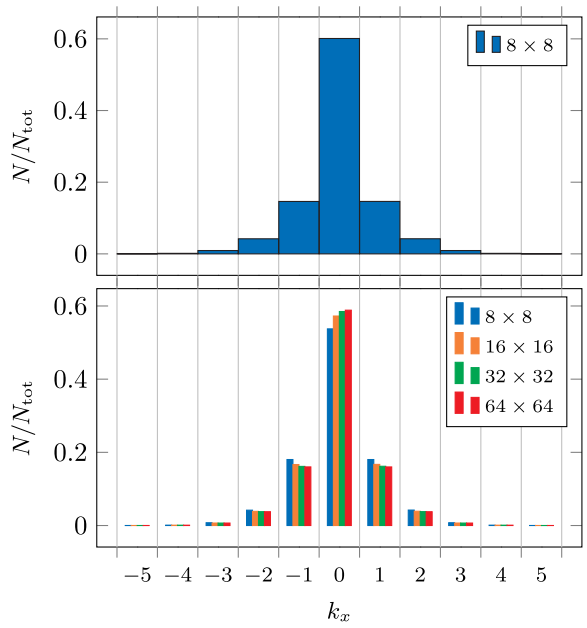


FIG. 19. Distributions of the link field k_x. These two histograms feature the distributions of k_x in the training set (top) and in the test set (bottom).

APPENDIX C: ADDITIONAL PROOFS

This Appendix contains proofs referenced in Sec. III. The idea of the first lemma is to show a simple, albeit arguably trivial, example of a network's prediction being invariant under translations of the input even though the network contains a layer that breaks translational equivariance before the global pooling layer.

Lemma 1.—Given an $N \times N'$ feature map f and a $k \times k$ spatial average pooling layer P with stride $s = k$, $k|N$, and $k|N'$, applying a global average pooling layer directly after the spatial average pooling layer is equivalent to applying only the global average pooling layer and omitting the spatial average pooling layer.

Proof.—We want to show that

$$\mathrm{GAP}(Pf(x)) = \mathrm{GAP}(f(y)). \tag{C1}$$

The global average pooling over an $N \times N'$ feature map f is given by

$$\mathrm{GAP}(f(y)) = \frac{1}{NN'}\sum_{y\in F} f(y), \tag{C2}$$

with

$$f: F \subset \mathbb{Z}^2 \to \mathbb{R}. \tag{C3}$$

The spatial average pooling P can be interpreted as a special convolutional layer

$$Pf(x) = [f \star \psi]_{s=k}(x) = \frac{1}{k^2}\sum_{\phi\in\Psi} f(kx+\phi), \tag{C4}$$

using the filter $\psi(x) = 1/k^2$, where

$$\psi: \Psi \subset \mathbb{Z}^2 \to \mathbb{R}. \tag{C5}$$

The resulting feature map $f': F' \subset \mathbb{Z}^2 \to \mathbb{R}$ has the dimensions $N/k \times N'/k$. The validity of Eq. (C1) can be seen by

$$\begin{aligned}\mathrm{GAP}(Pf(x)) &= \frac{1}{(N/k)(N'/k)}\sum_{x\in F'}\frac{1}{k^2}\sum_{\phi\in\Psi} f(kx+\phi)\\ &= \frac{1}{NN'}\sum_{y\in F} f(y)\\ &= \mathrm{GAP}(f(y)).\end{aligned} \tag{C6}$$

The first step uses Eqs. (C2) and (C4); note that the GAP is performed over the feature map f'. The second equality holds for $s = k$, $k|N$, and $k|N'$, and the last one utilizes again Eq. (C2) but for the feature map f. ∎

Remark.—Even though the spatial average pooling layer with $s > 1$ breaks translational equivariance under arbitrary translations, the result after the GAP is still invariant under translations.

The following lemma shows that a convolutional layer that is directly followed by a global average pooling layer does not have the effect that one might expect a regular convolution to have. Loosely speaking, it does not effectively increase the network's depth, because it collapses with the global pooling layer. This lemma should, therefore, also highlight the importance of the usage of an activation function between the last convolutional and the global average pooling layer.

Lemma 2.—Given an $M \times M'$ feature map f' and an $l \times l$ convolution ψ': $\Psi' \subset \mathbb{Z}^2 \to \mathbb{R}$ with a stride of one and a single output channel, applying the convolution and then performing a global average pooling is equivalent to performing the global average, multiplying it by the sum of the convolution's weights, and adding the bias b.

Proof.—What we want to show is

$$\mathrm{GAP}([f' \star \psi'](x) + b) = \mathrm{GAP}(f'(x))\sum_{\phi'\in\Psi'}\psi'(\phi') + b. \tag{C7}$$

This can be seen by

$$\begin{aligned}
&\mathrm{GAP}([f' \star \psi'](x) + b) \\
&= \frac{1}{MM'} \sum_{x \in F'} \left(\sum_{\phi' \in \Psi'} f'(x + \phi') \psi'(\phi') + b \right) \\
&= \frac{1}{MM'} \sum_{x \in F'} \sum_{\phi' \in \Psi'} f'(x + \phi') \psi'(\phi') + b \\
&= \sum_{\phi' \in \Psi'} \psi'(\phi') \frac{1}{MM'} \sum_{x \in F'} f'(x + \phi') + b \\
&= \sum_{\phi' \in \Psi'} \psi'(\phi') \mathrm{GAP}(f'(x)) + b \\
&= \mathrm{GAP}(f'(x)) \sum_{\phi' \in \Psi'} \psi'(\phi') + b. \qquad \text{(C8)}
\end{aligned}$$

The first equality combines the definitions of the GAP, given by Eq. (C2), and the convolution, given by Eq. (10). The second one utilizes the fact that the bias does not depend on the lattice site x. The third step takes advantage of the fact that ψ' does not depend on x, and the fourth one makes use of the periodic boundary conditions and of Eq. (C2). The last equality holds because the result of the GAP does not depend on ϕ'. ∎

Remark.—This is possible only without an activation function between the convolutional layer and the global average pooling. Also note the importance of periodic boundary conditions.

The following theorem combines both lemmas and shows that a spatial average pooling layer, followed by a convolutional and a global average pooling layer, still leads to an output that is invariant under translations of the input if the strides and kernel sizes are chosen appropriately. It emphasizes once again the importance of an activation function before the global average pooling.

Theorem 1.—Given an $N \times N'$ feature map, a $k \times k$ spatial average pooling layer with stride $s = k$, $k|N$, $k|N'$, and an $l \times l$ convolution ψ' with a stride of one and a single output channel, applying the spatial average pooling layer, then the convolution and then the global average pooling layer, is equivalent to applying the global average pooling layer, multiplying the result by the sum of the convolution's weights, and adding the bias b.

Proof.—Combining Lemmas 1 and 2 with $M = N/k$ and $M' = N'/k$ leads to the desired result. ∎

Remark.—The generalization to more than one feature map and multiple output channels is straightforward.

APPENDIX D: PARTIALLY OCCLUDED INPUT

In this Appendix, we analyze the worse performance of the ST models on the 100×5 lattice from Sec. IV, which is depicted in Fig. 5. The source of the problem is that a model ignores part of the data at a strided operation if the kernel and stride are not compatible with the data's shape that the layer in question receives. In the case of the ST models, these operations are the strided spatial pooling layers. To examine this problem in more detail, we perform an experiment in which we hide a portion of the input data from the network for both architectures, ST and EQ: On every lattice size, the network is shown only a part of the input. This is done by discarding 20% of it either on the right or on the bottom of the lattice, so that the resulting restricted input still has a rectangular shape and, thus, a valid input size for the networks. This way, only 80% of the input data are shown to the network, but the value of the observable still corresponds to the full lattice configuration. The input size of the 50×2 lattice becomes 40×2, 60×4 becomes 48×4, 100×5 becomes 100×4, 125×8 becomes 100×8, and 200×10 becomes 160×10 and 200×8, respectively. Note that the data are discarded four times in the temporal and twice in the spatial direction. The result of this experiment is shown in Fig. 20. The overall test loss (top) shows two kinks, namely, for the lattices, for which the input was restricted in the spatial direction. These kinks are also seen in the loss curve corresponding to $|\phi|^2$ (bottom) but are much more pronounced for n (middle). In fact, if the data are discarded in the temporal direction, the quality of the prediction of n barely changes at all, as can be seen by comparing the middle plot in Fig. 5 to the middle plot in

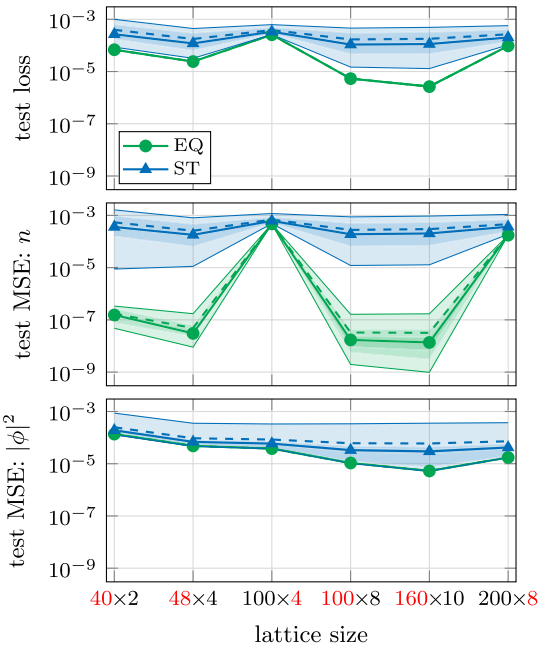


FIG. 20. Test loss (top) and its two parts (middle and bottom) that come from each observable, corresponding to discarding 20% of the input data, on various lattice sizes. The networks are trained on a 60×4 lattice. The dimension along which the input data are occluded is marked in red. For n, the predictions become worse if the data are concealed in the spatial direction. For $|\phi|^2$, the predictions become worse if any data are hidden, but they are slightly worse if data are suppressed in the spatial direction. The reason for this lies in the nature of the observables.

Fig. 20. The reason for this is the way the configurations are generated, namely, with the worm algorithm.

The nature of said algorithm is local, and each step involves adjacent points, giving rise to modifications of the field values that are contiguous on the lattice. This can be formally expressed by interpreting worms as paths on the toruslike space corresponding to the lattice with its periodic boundary conditions. For this task, we deal with only closed worm configurations, so the paths are, in fact, loops. With this picture in mind, Eq. (20) represents an (averaged) winding number in the temporal dimension of the torus. Discarding data in this direction does not alter the winding number. On the other hand, if data are discarded along the other dimension, parts of the worm might be discarded as well, leading to a very high discrepancy from the true winding number.

The small kinks in $|\phi|^2$ can be explained by means of some additional remarks: First of all, in the range of f_x in our dataset, the ratio $W(f_x+2)/W(f_x)$ is almost linear, so $|\phi|^2$ can be viewed as the average of f_x in first approximation. The functions W and f_x are given by Eqs. (A9) and (A10), respectively. The link variables l_t and l_x are not modified by the worm but by a standard Monte Carlo process; hence, their distribution is not biased in any direction, and the average over the truncated lattice has small deviations from the one over the whole lattice. Note that these deviations decrease as the lattice increases in size. Unlike the integer field k_t, k_x is not coupled to the chemical potential; therefore, it is less likely for worms to wind around the spatial dimension than the temporal. This means that the average of k_x is affected by a cut in the time dimension, but deviations from the true value do not occur often and decrease as the lattice size increases. Combining all these remarks finally yields the reason for those two kinks being at the same value of the restricted lattice size for both observables in Fig. 20 and the reason why they are less pronounced for $|\phi|^2$.

[1] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biol. Cybern. **36**, 193 (1980).

[2] K. Fukushima, Cognitron: A self-organizing multilayered neural network, Biol. Cybern. **20**, 121 (1975).

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. **115**, 211 (2015).

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM **60**, 84 (2017).

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE **86**, 2278 (1998).

[6] M. Lin, Q. Chen, and S. Yan, Network in network, arXiv: 1312.4400.

[7] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016* (IEEE, Las Vegas, USA, 2016), pp. 770–778.

[8] T. S. Cohen and M. Welling, Group equivariant convolutional networks, in *Proceedings of the 33rd International Conference on Machine Learning, Vol. 48, PMLR, 2016* (JMLR, New York, USA, 2016), pp. 2990–2999.

[9] T. S. Cohen and M. Welling, Steerable CNNs, in *Proceedings of the International Conference on Learning Representations (ICLR), 2017* (OpenReview, Toulon, France, 2017).

[10] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, Harmonic networks: Deep translation and rotation equivariance, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017* (IEEE, Honolulu, USA, 2017), pp. 7168–7177.

[11] D. Worrall and G. Brostow, CubeNet: Equivariance to 3D rotation and translation, in *Proceedings of the European Conference on Computer Vision (ECCV), 2018* (Springer, Cham; Munich, Germany, 2018), pp. 567–584.

[12] A. S. Ecker, F. H. Sinz, E. Froudarakis, P. G. Fahey, S. A. Cadena, E. Y. Walker, E. Cobos, J. Reimer, A. S. Tolias, and M. Bethge, A rotation-equivariant convolutional neural network model of primary visual cortex, in *Proceedings of the International Conference on Learning Representations (ICLR), 2019* (OpenReview, New Orleans, USA, 2019).

[13] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, and M. Welling, Rotation equivariant CNNs for digital pathology, in *Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2018* (Springer International Publishing, Granada, Spain, 2018), pp. 210–218.

[14] T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, Gauge equivariant convolutional networks and the icosahedral CNN, in *Proceedings of the 36th International Conference on Machine Learning, Vol. 97, PMLR, 2019* (JMLR, Long Beach, USA, 2019), pp. 1321–1330.

[15] M. W. Lafarge, E. J. Bekkers, J. P. W. Pluim, R. Duits, and M. Veta, Roto-translation equivariant convolutional networks: Application to histopathology image analysis, Med. Image Anal. **68**, 101849 (2021).

[16] S. Pang, A. Du, M. A. Orgun, Y. Wang, Q. Sheng, S. Wang, X. Huang, and Z. Yu, Beyond CNNs: Exploiting further inherent symmetries in medical images for segmentation, arXiv:2005.03924.

[17] K. A. Nicoli, C. J. Anders, L. Funcke, T. Hartung, K. Jansen, P. Kessel, S. Nakajima, and P. Stornati, Estimation

of Thermodynamic Observables in Lattice Field Theories with Deep Generative Models, Phys. Rev. Lett. **126**, 032001 (2021).

[18] A. M. M. Scaife and F. Porter, Fanaroff-riley classification of radio galaxies using group-equivariant convolutional neural networks, Mon. Not. R. Astron. Soc. **503**, 2369 (2021).

[19] G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racanière, D. J. Rezende, and P. E. Shanahan, Equivariant Flow-Based Sampling for Lattice Gauge Theory, Phys. Rev. Lett. **125**, 121601 (2020).

[20] D. Boyda, G. Kanwar, S. Racanière, D. J. Rezende, M. S. Albergo, K. Cranmer, D. C. Hackett, and P. E. Shanahan, Sampling using $SU(N)$ gauge equivariant flows, Phys. Rev. D **103**, 074504 (2021).

[21] M. Favoni, A. Ipp, D. I. Müller, and D. Schuh, Lattice gauge equivariant convolutional neural networks, arXiv:2012.12901.

[22] K. Zhou, G. Endrődi, L.-G. Pang, and H. Stöcker, Regressive and generative neural networks for scalar field theory, Phys. Rev. D **100**, 011501 (2019).

[23] S. J. Wetzel and M. Scherzer, Machine learning of explicit order parameters: From the Ising model to SU(2) lattice gauge theory, Phys. Rev. B **96**, 184410 (2017).

[24] D. Bachtis, G. Aarts, and B. Lucini, Mapping distinct phase transitions to a neural network, Phys. Rev. E **102**, 053306 (2020).

[25] D. Bachtis, G. Aarts, and B. Lucini, Extending machine learning classification capabilities with histogram reweighting, Phys. Rev. E **102**, 033303 (2020).

[26] S. Blücher, L. Kades, J. M. Pawlowski, N. Strodthoff, and J. M. Urban, Towards novel insights in lattice field theory with explainable machine learning, Phys. Rev. D **101**, 094507 (2020).

[27] K. Padavala, A. Singh, and J. Kundu, Machine learned phase transitions in a system of anisotropic particles on a square lattice, arXiv:2102.03006.

[28] Y. Wang, Z. Cao, and A. B. Farimani, Deep reinforcement learning optimizes graphene nanopores for efficient desalination, arXiv:2101.07399.

[29] K. Zhang, S. Lederer, K. Choo, T. Neupert, G. Carleo, and E.-A. Kim, Hamiltonian reconstruction as metric for variational studies, arXiv:2102.00019.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2019* (Association for Computing Machinery, New York, NY, USA; Anchorage, AK, USA, 2019), pp. 2623–2631.

[31] E. Noether, Invariante Variationsprobleme, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Math.-Phys. Kl. **1918**, 235 (1918), http://eudml.org/doc/59024.

[32] C. Gattringer and T. Kloiber, Lattice study of the Silver Blaze phenomenon for a charged scalar ϕ^4 field, Nucl. Phys. **B869**, 56 (2013).

[33] N. Prokof'ev and B. Svistunov, Worm Algorithms for Classical Statistical Models, Phys. Rev. Lett. **87**, 160601 (2001).

[34] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. **2**, 303 (1989).

[35] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, The expressive power of neural networks: A view from the width, Adv. Neural Inf. Process. Syst. **30** (2017), https://papers.nips.cc/paper/2017/hash/32cbf687880eb1674a07bf717761dd3a-Abstract.html.

[36] A. Paszke *et al.*, PyTorch: An imperative style, high-performance deep learning library, Adv. Neural Inf. Process Syst. **32** (2019), https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[37] F. Yu and V. Koltun, Multi-scale context aggregation by dilated convolutions, in *Proceedings of the International Conference on Learning Representations (ICLR), 2016* (OpenReview, San Juan, Puerto Rico, 2016).

[38] A. L. Maas, A. Y. Hannun, and A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013* (ICML, Atlanta, USA, 2013), https://sites.google.com/site/deeplearningicml2013/relu_hybrid_icml2013_final.pdf.

[39] S. J. Reddi, S. Kale, and S. Kumar, On the convergence of Adam and beyond, in *Proceedings of the International Conference on Learning Representations (ICLR), 2018* (ICLR, Scottsdale, USA, 2018).

[40] I. Loshchilov and F. Hutter, Fixing weight decay regularization in Adam, in *Proceedings of the International Conference on Learning Representations (ICLR), 2019* (OpenReview, New Orleans, USA, 2019).

[41] T. D. Cohen, Functional Integrals for QCD at Nonzero Chemical Potential and Zero Density, Phys. Rev. Lett. **91**, 222001 (2003).

[42] C. Gattringer and T. Kloiber, Spectroscopy in finite density lattice field theory: An exploratory study in the relativistic bose gas, Phys. Lett. B **720**, 210 (2013).

[43] T. Rindlisbacher, O. Åkerlund, and P. de Forcrand, Sampling of general correlators in worm-algorithm based simulations, Nucl. Phys. **B909**, 542 (2016).

[44] G. Gao, J. Gao, Q. Liu, Q. Wang, and Y. Wang, CNN-based density estimation and crowd counting: A survey, arXiv:2003.12783.

[45] See https://gitlab.com/openpixi/scalar_ml.