Scattering with neural operators

Sebastian Mizera●

Institute for Advanced Study, Einstein Drive, Princeton, New Jersey 08540, USA

Recent advances in machine learning establish the ability of certain neural-network architectures called neural operators to approximate maps between function spaces. Motivated by a prospect of employing them in fundamental physics, we examine applications to scattering processes in quantum mechanics. We use an iterated variant of Fourier neural operators to learn the physics of Schrödinger operators, which map from the space of initial wave functions and potentials to the final wave functions. These deep operator learning ideas are put to test in two concrete out-of-distribution problems: a neural operator predicting the time evolution of a wave packet scattering off a central potential in $1+1$ dimensions, and the double-slit experiment in $2+1$ dimensions. At inference, neural operators can become orders of magnitude more efficient compared to traditional finite-difference solvers.

Do androids dream of Schrödinger's cats? Recent advances in machine learning demonstrated the ability of neural networks to approximate not only functions, but also nonlinear operators, using various architectures collectively known as neural operators [1–5]. In this work, we ask whether they can serve as a practical computational tool in fundamental physics.[1] This question is motivated in part by the exploding complexity of perturbative quantum field theory computations needed for precision predictions at the Large Hadron Collider [9,10], calling for rethinking how to represent and compute S-matrix elements efficiently.

This investigation has to start somewhere and here we consider arguably the simplest scattering operator in $(d+1)$-dimensional quantum mechanics,

$$S[V(\vec{x})] = \mathcal{T} e^{-i/\hbar \int_0^T \hat{H}[V(\vec{x})]dt}, \qquad (1)$$

which acts on the initial position-space wave function $\Psi(\vec{x}, 0)$ to produce the final one $\Psi(\vec{x}, T)$ at some fixed time T:

$$\Psi(\vec{x}, T) = S[V(\vec{x})]\Psi(\vec{x}, 0). \qquad (2)$$

Here, $(\vec{x}, t)$ are the space-time coordinates, $\mathcal{T}$ is the time-ordering operator, and $\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + gV(\vec{x})$ is the

time-independent Hamiltonian, where m is the particle mass, and g is a coupling constant. We emphasized the functional dependence on the potential V, which is precisely what makes S a nonlinear operator, viewed as acting simultaneously on the space of Vs and initial conditions for Ψ.

We ask whether, instead of computing the time evolution using traditional methods, the Schrödinger operator S can be represented as a neural operator $\mathcal{N}$ (defined more precisely below), so that

$$\Psi(\vec{x}, T) \stackrel{?}{=} \mathcal{N}[V(\vec{x}), \Psi(\vec{x}, 0)]. \qquad (3)$$

The universal approximation theorem for operators, originally due to Chen and Chen [11], actually guarantees an affirmative answer. Translated to our setup, it implies that for any $\epsilon > 0$, there exists a complicated enough $\mathcal{N}_\epsilon$ such that the L^2 norm $\|\Psi(\vec{x}, T) - \mathcal{N}_\epsilon[V(\vec{x}), \Psi(\vec{x}, 0)]\| < \epsilon$ for every continuous $V(\vec{x})$ and $\Psi(\vec{x}, 0)$, see [12–14] for details. We will make this statement more precise shortly.

Nevertheless, it is not known how to find such an $\mathcal{N}_\epsilon$ constructively (it would be akin to looking for the next best-selling novel in the decimal expansion of π). Indeed, constructing decent $\mathcal{N}$s became practically viable only recently, by combining neural operators with deep learning ideas [1,4]. In this work, we examine whether this framework allows us to machine learn the Schrödinger operator in a meaningful way.

Strategy: Let us sketch the general idea first. The neural operator is trained on triples of data:

$$\underbrace{V(\vec{x}), \Psi(\vec{x}, 0)}_{\text{input}}, \underbrace{\Psi(\vec{x}, T)}_{\text{output}}, \qquad (4)$$

---

[1]Neural networks, but not operators, have already had transformative impact on particle-physics phenomenology and beyond, see, e.g., [6–8].
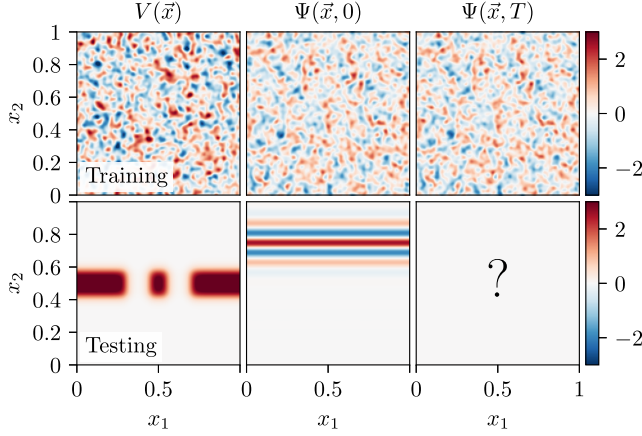
FIG. 1. Example training (top) and testing (bottom) data with the input in the first two columns and the output in the third one. Only real parts are displayed for conciseness.

TABLE I. Comparison between single time step GPU times (in seconds) and memory usage (in GB) of the Crank-Nicholson (CN) method and neural operator (NO) for different dimensions d and lattice sizes N. The last column gives an average L^2-error between the two methods across the testing dataset. Original training sizes for NO are bold. Crosses indicate that a computation did not terminate due to memory shortage.

d	N	CN time	CN memory	NO time	NO memory	Error
1	**256**	0.013	0.004	0.002	0.13	0.0005
1	2048	0.2	0.3	0.003	0.14	0.004
1	16384	56	20	0.003	0.2	0.004
1	32768	✗	>40	0.003	0.3	✗
2	**64**	0.5	0.6	0.005	0.16	0.0011
2	128	22	10	0.005	0.2	0.03
2	256	✗	>40	0.005	0.4	✗

where the input is a Gaussian noise and $\Psi(\vec{x}, T)$ is computed exactly according to (2), see Fig. 1 (top). The learning process is entirely data driven and has no physics input: $\mathcal{N}$ has to effectively "dream up" its own representation of the Schrödinger equation from scratch. In fact, the neural operator learns not just one instance of this equation, but instead a whole family parameterized by the interaction potential V.

It would not be surprising if $\mathcal{N}$ could accurately solve problems drawn from the same distribution as it was trained on. Therefore, after training, or at inference, $\mathcal{N}$ is instead tasked with predicting the time evolution of a previously unseen problem: for example, a wave packet approaching a double-slit potential, see Fig. 1 (bottom). In other words, we are probing the generalization or out-of-distribution error: the ability of neural operators to extrapolate the information learned during training to new problems.

The prediction needs to be accurate enough to be able to iterate this process k times and study strongly coupled scattering by evolving the wave function $\Psi(\vec{x}, kT)$ for long times. As we will see in Table I, at inference, this way of encoding physics can become faster and more memory efficient compared to traditional algorithms, such as finite-difference methods, because once the operator $\mathcal{N}$ is learned, it does not need to be recomputed for every new input V and Ψ.

Architecture. We use a version of the Fourier neural operator architecture [3] adapted to the problem at hand.

Channels: The data is organized into channels. We take the input $\mathbf{I}: D \to \mathbb{R}^{d+3}$ to consist of $d + 3$ channels,

$$\mathbf{I}(\vec{x}) = \big(V(\vec{x}), \mathrm{Re}\Psi(\vec{x}, 0), \mathrm{Im}\Psi(\vec{x}, 0), \vec{x}\big), \qquad (5)$$

where $D \ni \vec{x}$ is the spatial domain we take to be the unit torus $D = \mathbb{T}^d$ (we impose periodic boundary conditions for simplicity). In practice, one samples points from a

discretization of D, such as a uniform lattice of N^d points. Likewise, the output $\mathbf{O}: D \to \mathbb{R}^2$ consists of two channels:

$$\mathbf{O}(\vec{x}) = \big(\mathrm{Re}\Psi(\vec{x}, T), \mathrm{Im}\Psi(\vec{x}, T)\big). \qquad (6)$$

Splitting Ψs into real/imaginary parts and adding positional embeddings $\vec{x}$ is used to increase numerical stability and facilitate the learning process.

The architecture will be designed to be independent of the specific discretization of D, which in particular means that a neural operator trained on a coarse lattice can be used to make predictions on a finer one, as will be discussed below.

Neural operator: A neural operator $\mathcal{N}$ is written as a composition of $\mathrm{L} + 2$ layers resembling standard deep neural networks [4]:

$$\mathcal{N} = \mathcal{P} \circ f_{\mathrm{L}} \circ \cdots \circ f_2 \circ f_1 \circ \mathcal{L}. \qquad (7)$$

Here, $\mathcal{L}$ and $\mathcal{P}$ are the lifting and projection layers, respectively, which act as identities in D and whose role is to map the input data $\mathbf{I}$ to its hidden representations with h channels, $\mathbf{H}_\ell: D \to \mathbb{R}^h$ for $\ell = 0, 1, \ldots, \mathrm{L}$, and then back to the output $\mathbf{O}$:

$$\mathbf{I} \overset{\mathcal{L}}{\mapsto} \mathbf{H}_0 \overset{f_1}{\mapsto} \mathbf{H}_1 \overset{f_2}{\mapsto} \cdots \overset{f_{\mathrm{L}}}{\mapsto} \mathbf{H}_{\mathrm{L}} \overset{\mathcal{P}}{\mapsto} \mathbf{O}. \qquad (8)$$

For us, $\mathcal{L}$ and $\mathcal{P}$ are implemented as one- and two-layer perceptrons, respectively, with the hidden dimension p in the second case. Typically, $\mathrm{L} = 4$ and $h, p \gg d$. The hidden-layer functions $f_{\ell+1}$ are defined through

$$\mathbf{H}_{\ell+1}(\vec{x}) = \sigma\big(W_\ell \mathbf{H}_\ell(\vec{x}) + (\mathcal{K}_\ell \mathbf{H}_\ell)(\vec{x}) + b_\ell\big) + s_\ell \mathbf{H}_\ell(\vec{x}), \qquad (9)$$

where the weights $W_\ell \in \mathbb{R}^{h \times h}$, biases $b_\ell \in \mathbb{R}^h$, and skip connections $s_\ell \in \mathbb{R}$ act as identities in D. The activation

function σ is the only nonlinearity and is applied element wise. We take it to be the Gaussian error linear unit [15].

Fourier layers: The key objects are the (linear) integral kernel operators $\mathcal{K}_\ell$ with $\ell = 1, 2, ..., L$, which take the general form

$$(\mathcal{K}_\ell \mathbf{H}_\ell)(\vec{x}) = \int_D K_\ell(\vec{x}, \vec{y}) \mathbf{H}_\ell(\vec{y}) \mathrm{d}^d \vec{y} \qquad (10)$$

for all $\vec{x} \in D$. Note this is the only layer acting non-diagonally in D.

Out of the previously explored choices for parametrizing the convolution kernel K_ℓ [2–4], we write (10) using Fourier transforms $\mathcal{F}$ and their inverses $\mathcal{F}^{-1}$:

$$\mathcal{K}_\ell \mathbf{H}_\ell = \mathcal{F}^{-1}\big[\mathcal{F}[K_\ell]\mathcal{F}[\mathbf{H}_\ell]\big]. \qquad (11)$$

This choice allows us to represent $\mathcal{F}[K_\ell]$ directly in Fourier space $D^* \ni \vec{p}$, as a tensor $\mathbf{T}_\ell \in \mathbb{C}^{F \times h \times h}$, where F is the number of Fourier modes. The expression (11) can then be evaluated efficiently using fast Fourier transforms and tensor multiplication.

To summarize, the parameters of $\mathcal{N}$ are the entries of W_ℓ, $\mathbf{T}_\ell$, b_ℓ, and s_ℓ that we will optimize for, see Fig. 2. The hyperparameters are all the numbers describing the network properties, such as L, h, F, etc. In this case, the afore-mentioned universal approximation theorem states that one can always find large enough values of the hyperparameters such that there exist specific W_ℓ, $\mathbf{T}_\ell$, b_ℓ, and s_ℓ for which the L^2 error is bounded by ϵ for any input in a compact subset of a Banach space [12–14]. Note that the theorem would not be true if we did not include the nonlinearity σ and the Fourier layers.

In practice, we truncate the number of modes to be half along each dimension, i.e., $F = (N/2)^d$, and ensure reality of the output by imposing $\mathcal{F}[K_\ell](\vec{p}) = \mathcal{F}[K_\ell](-\vec{p})^*$ on the remaining ones. Since the tensors $\mathbf{T}_\ell$ are by far the largest part of the neural operator (contributing $2Lmh^2$ real parameters), it pays off to instead represent them using Tucker decomposition with rank, or compression ratio, r [16].

We have tried adding a number of other features, including normalization layers, dropout regularization [17], and Sobolev training [18] without drastic improvements in the
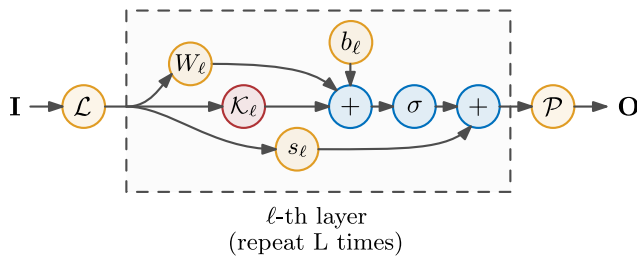

ℓ-th layer
(repeat L times)

FIG. 2. Diagram of the neural operator architecture used in this work. The Fourier layer (red) is introduced in (11).

results. Nonetheless, we suspect they will be important in large-scale applications of neural operators.

There exist different open-source implementations of Fourier neural operator [19], as well as other neural-operator architectures, including DeepONet [1] and their variations [20–37].

Loss function: We use the L^2 error as the loss function:

$$\text{loss} = \big\|\Psi(\vec{x}, T) - \mathcal{N}[V(\vec{x}), \Psi(\vec{x}, 0)]\big\|, \qquad (12)$$

where $\Psi(\vec{x}, T)$ is the answer computed using the Crank-Nicolson finite difference method, which for our purposes will be referred to as exact or as the ground truth below. To avoid confusion, we will add subscripts: $\Psi_{\text{GT}}(\vec{x}, T) = \Psi(\vec{x}, T)$ and $\Psi_{\text{NO}}(\vec{x}, T) = \mathcal{N}[V(\vec{x}), \Psi(\vec{x}, 0)]$ to distinguish between the exact and inferred computations.

We use the AdamW optimizer [38], which is an adaptive variant of stochastic gradient descent, with learning rate ν and weight decay w, together with a scheduler halving ν every time the training loss reaches a plateau.

Training dataset: The training set is constructed by drawing the first three entries of (5) independently from a Gaussian random field with spatial width $\mu = 0.1$ (i.e., the power spectrum $\propto e^{-\mu^2|\vec{p}|^2/2}$), sampled over a uniform grid of N^d points on $\mathbb{T}^d$. In particular, the potential $V(\vec{x})$ is purely real. In order to probe unitarity, each sample is normalized such that $\|\Psi(\vec{x}, 0)\| = 1$ and we similarly set $\|V(\vec{x})\| = 1$. We then create a timeline by iteratively solving for $\Psi_{\text{GT}}(\vec{x}, kT)$ up to $k \leq M = 256$. Hence, each timeline produces $M - 1$ input-output pairs. For numerical stability, we normalize each output such that $\|\Psi_{\text{GT}}(\vec{x}, kT)\| = 1$ before evolving it to $\Psi_{\text{GT}}(\vec{x}, (k+1)T)$. We repeat this process $K = 32$ times, which results in the training dataset of size $n_{\text{train}} = K(M-1) = 8160$. See Figs. 1 and 3 (top rows) for examples.

We emphasize that it is possible to achieve much better performance with a training set adapted to scattering problems, for example by including Ψs resembling wave packets. Likewise, one can include probability conservation in the loss function, hard code the fact $\mathcal{N}$ is supposed to be linear in Ψ, or make use of other physics-informed training strategies [39–42]. Here, we purposely do not employ these steps, because our goal is to probe the out-of-distribution error and find out whether $\mathcal{N}$ has learned quantum physics.

All computations are performed on a single NVIDIA A100 Graphics Processing Unit (GPU) with 40 GB memory.

Wave packet scattering. As the first application, we consider wave packet scattering in $d = 1$. We set $\hbar = m = 1$ and $g = 3 \times 10^3$. In those units, the time step is chosen to be $T = 6.3 \times 10^{-5}$, which is fine enough to produce a "movie," see [43]. Much smaller Ts can cause numerical approximation errors and much larger ones prove more
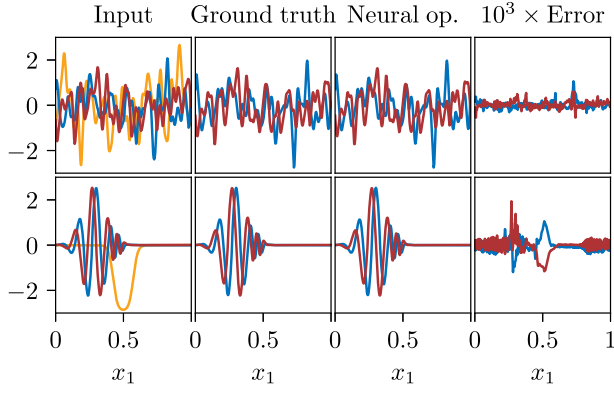
FIG. 3. Comparison between the exact and inferred wave functions for training (top) and testing (bottom) datasets in $d = 1$. From left to right: potential $V(x_1)$ (yellow) and the initial wave function $\Psi(x_1, 0)$ (real part in red, imaginary part in blue); exact wave function $\Psi_{GT}(x_1, T)$; neural operator output $\Psi_{NO}(x_1, T)$; and the error $\Psi_{GT}(x_1, T) - \Psi_{NO}(x_1, T)$ magnified $10^3\times$ for visibility. The difference between $t = 0$ and $t = T$ is nonzero but barely visible by eye due to the choice of a small time step; see Fig. 6 for time evolution across $t \leq 256T$.

difficult to train. The spatial dimension is a unit circle $\mathbb{T}^1 \ni x_1$ and we discretize it with $N = 256$ points.

Testing dataset: The testing dataset is prepared in exactly the same way as the training one, except it is out of distribution: we start with the specific wave packet $\Psi(x_1, 0)$ approaching a potential well $V(x_1)$, as illustrated in Fig. 3 (bottom left). After evolving it through time $t \leq MT$, the testing dataset comprises of $n_{\text{test}} = M - 1 = 255$ samples.

Hyperparameters: We performed a Bayesian search over the hyperparameter space, which revealed preference for relatively small networks that are less prone to overfitting. In the results below, we use $h = 64$, $p = 512$ hidden and projection channels, $L = 4$ Fourier layers, and Tucker rank $r = 10^{-2}$. Learning rate is set to $\nu = 10^{-3}$, weight decay to $w = 10^{-5}$, and training is performed in batches of size $b = 32$. The resulting model has $\sim 9 \times 10^4$ parameters.

Example learning curves are shown in Fig. 4, where we also keep track of the testing loss as a benchmark. At first, it closely follows the training loss, but around epoch $\gtrsim 500$ the neural operator starts overtraining (memorizing instead of learning). We still find it beneficial to continue learning until both losses stabilize with the training loss converging to $\sim 3 \times 10^{-4}$ and testing to $\sim 5 \times 10^{-4}$. The kinks in the losses occur in places where the learning rate ν gets halved. Example training and testing data samples are compared in Fig. 3. Note that in the latter case, the wave function has flat near-zero segments, which are very nongeneric from the perspective of the training data, but nevertheless correctly evolved.

Unitarity: We found that unitarity can be treated as a proxy for the confidence of the neural operator about its predictions. In Fig. 5 (left), we display a histogram of
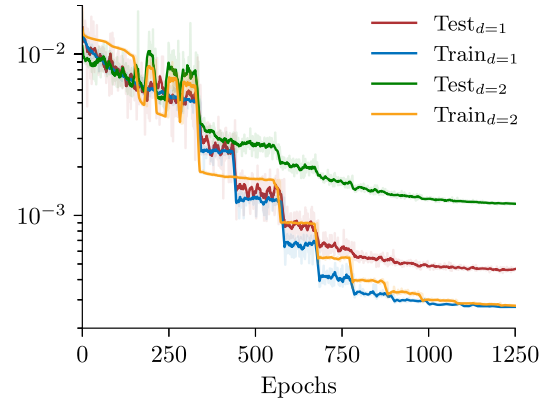

FIG. 4. Example training and testing losses for $d = 1, 2$. Single epoch takes ~ 1.8 ($d = 1$) and ~ 4.9 seconds ($d = 2$).

$\|\Psi_{NO}(x_1, T)\|$ over the whole training and testing datasets. It demonstrates the neural operator has learned unitarity with $\sim 10^{-4}$ precision.

Long-term predictions: Finally, we discuss the ability of the neural operator to make long-term scattering predictions by iterating $\mathcal{N}$,

$$\Psi_{NO}(\vec{x}, kT) = \mathcal{N}^k[V(\vec{x}), \Psi(\vec{x}, 0)], \qquad (13)$$

where $k \leq M = 256$. For simplicity of exposition, we keep the input potential V constant, even though the same $\mathcal{N}^k$ is also capable of treating time-dependent Vs. As with the finite-difference methods, we normalize $\|\Psi_{NO}(\vec{x}, kT)\| = 1$ after each iteration. Example timelines of $\|\Psi_{NO}(\vec{x}, t)\|$ before normalization are shown for $t \leq 256T$ in Fig. 5 (right) for examples of training and testing samples. They remain accurate to within $\sim 10^{-4}$ of identity, indicating high confidence in reliable predictions.

A full timeline is presented in Fig. 6, which shows the wave packet approaching the central potential, interfering with it, and emerging on the other side with a time delay and a spreading effect. The error builds up over time, but stays within ~ 0.02 even in the final time step $t = 256T$, showing strong generalization capabilities. It demonstrates
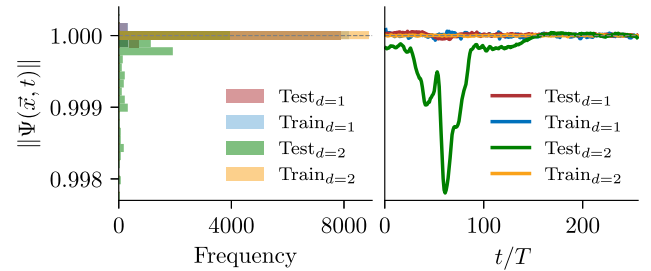

FIG. 5. Unitarity of the neural operator outputs for the training and testing datasets in $d = 1, 2$. Left: histogram of $\|\Psi_{NO}(\vec{x}, t)\|$ after one step $t = T$. Right: norms $\|\Psi_{NO}(\vec{x}, t)\|$ along the timeline $t \leq 256T$.
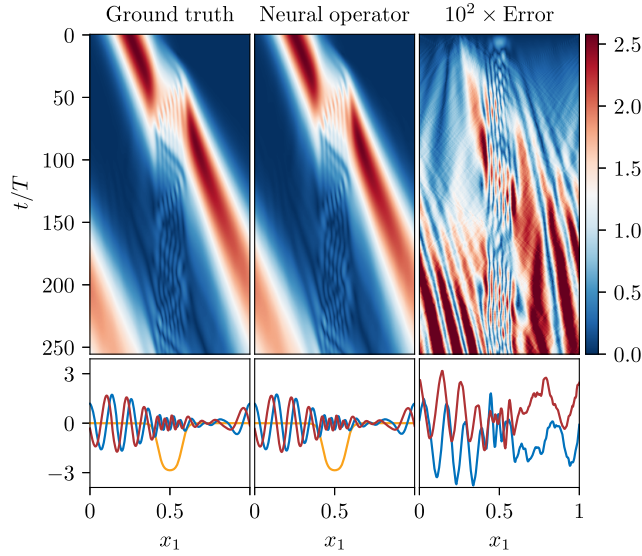
FIG. 6. Wave packet scattering in $d = 1$. Top: absolute values of the exact computation $|\Psi_{GT}(x_1, t)|$, the neural operator output $|\Psi_{NO}(x_1, t)|$, and the error $|\Psi_{GT}(x_1, t) - \Psi_{NO}(x_1, t)|$ multiplied by 10^2. Bottom: final-step wave function at $t = 256T$ in the same notation as in Fig. 3.

that the iterated neural operator $\mathcal{N}^k$ has effectively learned the physics of the Schrödinger operator at strong coupling in $d = 1$.

Double-slit experiment. We next discuss neural operators in $d = 2$, with the out-of-distribution test problem being a wave packet scattering off a double-slit potential, see Fig. 1 (bottom) for the initial conditions.

The hyperparameters used are the same as in the $d = 1$ case, except for the Tucker rank, which we take to be $r = 10^{-3}$ and the grid size, $N = 64$. The resulting number of parameters is $\sim 1.2 \times 10^5$. Training and testing dataset sizes remain the same as in $d = 1$.

Learning: As expected, the neural operator becomes more difficult to train in $d = 2$, as exemplified by the learning and unitarity curves in Figs. 4 and 5, respectively. The generalization error is worse by around an order of magnitude compared to $d = 1$: the unitarity errors stay within $\sim 4 \times 10^{-3}$ of identity; the training and testing losses reach $\sim 3 \times 10^{-4}$ and $\sim 1.1 \times 10^{-3}$, respectively. Unitarity dips around the time when the wave function develops high-frequency modes due to encountering the potential barrier.

Performance: As before, we construct $\mathcal{N}^k$ to study time evolution of the wave function. The results are illustrated in Fig. 7. After passing through the double-slit openings, the wave packet interferes with itself, creating a fringe pattern in the x_1 direction. Due to periodic boundary conditions, it also develops interference in the x_2 direction at late times. As in the $d = 1$ case, the numerical errors build up over time: in the first and last time frames, $t = T$ and $t = 256T$,
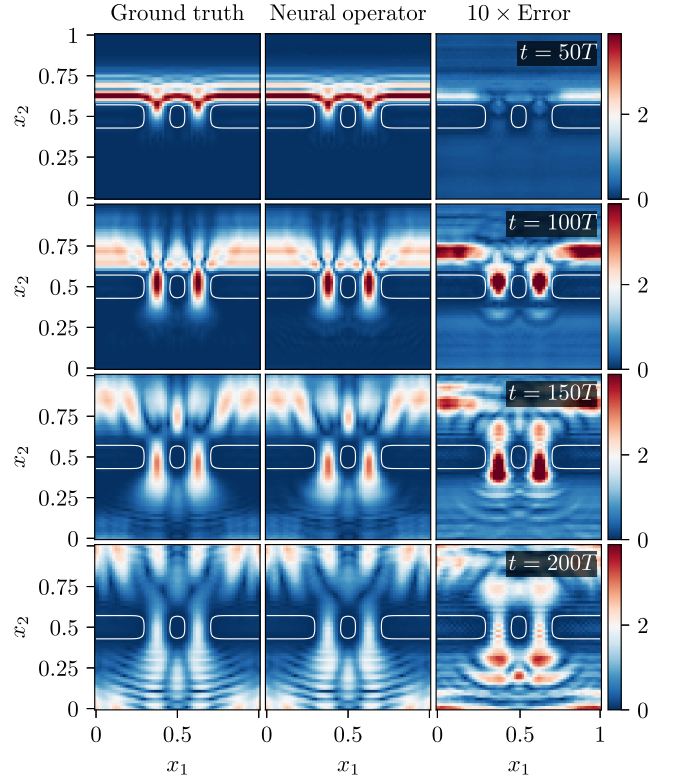

FIG. 7. Double-slit experiment in $d = 2$. From left to right: absolute values $|\Psi_{GT}(\vec{x}, t)|$, $|\Psi_{NO}(\vec{x}, t)|$, and $10 \times |\Psi_{GT}(\vec{x}, t) - \Psi_{NO}(\vec{x}, t)|$ for the indicated times t. An outline of the potential is plotted with white lines.

the L^2 error between the exact and inferred computations is $\sim 5 \times 10^{-4}$ and ~ 0.14, respectively.

Zero-shot super-resolution: Let us finish this discussion by emphasizing the distinction between a neural network acting on a very large but finite-dimensional Hilbert space (obtained by a specific discretization of D) and a neural operator, which acts on an infinite-dimensional space (independent of the discretization). In particular, it means that the latter can be trained on a lower-resolution lattice and then applied to computing $\Psi_{NO}(\vec{x}, T)$ at higher resolution. In the literature, this procedure is called a zero-shot super-resolution [4].

To exemplify it, in Table I we collected indicative times for computing $\Psi_{GT}(\vec{x}, T)$ and $\Psi_{NO}(\vec{x}, T)$ from the testing set at various resolutions, together with the L^2 errors between the two techniques. Finite-difference methods scale badly at large lattice sizes because they involve computing and taking powers of an $N^d \times N^d$ matrix for every new potential V. On the other hand, neural operator trained on the original sizes (here, $N = 256$ in $d = 1$ and $N = 64$ in $d = 2$) can be applied to finer grids and new potentials with low overhead.

Discussion. In this work, we explored the idea of using deep operator learning as a computational tool for mapping between function spaces appearing in fundamental physics.

As an illustrative example, we considered neural operators predicting quantum-mechanical scattering of wave packets with potential barriers and wells. We envisage that neural operators have much broader applicability in quantum theory, beyond just time-evolution operators, both in numerical and symbolic manipulations.

Let us emphasize that this strategy depends on a reliable way of producing training samples. The prospect is that, once trained, a neural operator can perform the same computation approximately but much more efficiently. The biggest open question concerns out-of-distribution errors. For example, given that neural operators represent physics in unconventional ways, would cheaply obtainable training data (e.g., coming from exactly solvable systems) suffice to learn solutions to conventionally difficult problems? We leave a study of this provocative question until future work.

[1] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Natl. Mach. Intell. **3,** 218 (2021).

[2] A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, and A. Stuart, Neural operator: Graph Kernel network for partial differential equations, in *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations* (2019), arXiv:2003.03485.

[3] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations, in *International Conference on Learning Representations* (2021), arXiv:2010.08895.

[4] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, Neural operator: Learning maps between function spaces, arXiv:2108.08481.

[5] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, Comput. Methods Appl. Mech. Eng. **393,** 114778 (2022).

[6] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Machine learning and the physical sciences, Rev. Mod. Phys. **91,** 045002 (2019).

[7] G. Karagiorgi, G. Kasieczka, S. Kravitz, B. Nachman, and D. Shih, Machine learning in the search for new fundamental physics, arXiv:2112.03769.

[8] S. Badger *et al.*, Machine learning and LHC event generation, SciPost Phys. **14,** 079 (2023).

[9] F. Caola, W. Chen, C. Duhr, X. Liu, B. Mistlberger, F. Petriello, G. Vita, and S. Weinzierl, The path forward to N^{3}LO, in *Snowmass 2021* (2022), arXiv:2203.06730.

[10] F. Febres Cordero, A. von Manteuffel, and T. Neumann, Computational challenges for multi-loop collider phenomenology: A Snowmass 2021 white paper, Comput. Softw. Big Sci. **6,** 14 (2022).

[11] T. Chen and H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Networks **6,** 911 (1995).

[12] S. Lanthaler, S. Mishra, and G. E. Karniadakis, Error estimates for DeepONets: A deep learning framework in infinite dimensions, Trans. Math. Appl. **6,** tnac001 (2022).

[13] N. Kovachki, S. Lanthaler, and S. Mishra, On universal approximation and error bounds for Fourier neural operators, J. Mach. Learn. Res. **22** (2021).

[14] T. D. Ryck and S. Mishra, Generic bounds on the approximation error for physics-informed (and) operator learning, arXiv:2205.11393.

[15] D. Hendrycks and K. Gimpel, Gaussian error linear units (GELUs), arXiv:1606.08415.

[16] J. Kossaifi, N. B. Kovachki, K. Azizzadenesheli, and A. Anandkumar, Multi-grid tensorized Fourier neural operator for high resolution PDEs (2023), https://openreview.net/forum?id=po-oqRst4Xm.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. **15,** 1929 (2014).

[18] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu, Sobolev training for neural networks, Adv. Neural Inf. Process. Syst. **30** (2017).

[19] GitHub repositories `neuraloperator`, `lululxvi/deeponet`, `SciML/NeuralOperators.jl`, `pde-bench/PDEBench`, `lu-group/deeponet-fno`, `HaoZhongkai/GNOT`, `scaomath/galerkin-transformer`, `ashiq24/UNO` (2023).

[20] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Multipole graph neural operator for parametric partial differential equations, arXiv:2006.09535.

[21] G. Gupta, X. Xiao, and P. Bogdan, Multiwavelet-based operator learning for differential equations, arXiv:2109.13459.

[22] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, Model reduction and neural networks for parametric PDEs, arXiv:2005.03180.

[23] J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro, Adaptive Fourier neural operators: Efficient token mixers for transformers, arXiv:2111.13587.

[24] S. Cao, Choose a transformer: Fourier or Galerkin, arXiv:2105.14995.

[25] M. A. Rahman, Z. E. Ross, and K. Azizzadenesheli, U-NO: U-shaped neural operators, arXiv:2204.11127.

[26] V. Fanaskov and I. Oseledets, Spectral neural operators, arXiv:2205.10573.

[27] P. Jin, S. Meng, and L. Lu, MIONet: Learning multiple-input operators via tensor product, SIAM J. Sci. Comput. **44**, A3490 (2022).

[28] M. Zhu, S. Feng, Y. Lin, and L. Lu, Fourier-DeepONet: Fourier-enhanced deep operator networks for full waveform inversion with improved accuracy, generalizability, and robustness, Comput. Methods Appl. Mech. Eng. **416**, 116300 (2023).

[29] Z. Jiang, M. Zhu, D. Li, Q. Li, Y. O. Yuan, and L. Lu, Fourier-MIONet: Fourier-enhanced multiple-input neural operators for multiphase modeling of geological carbon sequestration, arXiv:2303.04778.

[30] S. Wang, H. Wang, and P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONet, Sci. Adv. **7**, eabi8605 (2021).

[31] L. Lu, R. Pestourie, S. G. Johnson, and G. Romano, Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport, Phys. Rev. Res. **4**, 023210 (2022).

[32] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis, DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. **436**, 110296 (2021).

[33] M. Zhu, H. Zhang, A. Jiao, G. E. Karniadakis, and L. Lu, Reliable extrapolation of deep neural operators informed by physics or sparse observations, Comput. Methods Appl. Mech. Eng. **412**, 116064 (2023).

[34] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar, Fourier neural operator with learned deformations for PDEs on general geometries, arXiv:2207.05209.

[35] B. Deng, Y. Shin, L. Lu, Z. Zhang, and G. E. Karniadakis, Convergence rate of DeepONets for learning operators arising from advection-diffusion equations, arXiv:2102.10621.

[36] O. Ovadia, E. Turkel, A. Kahana, and G. E. Karniadakis, DiTTO: Diffusion-inspired temporal transformer operator, arXiv:2307.09072.

[37] M. A. Rahman, M. A. Florez, A. Anandkumar, Z. E. Ross, and K. Azizzadenesheli, Generative adversarial neural operators, arXiv:2205.03017.

[38] I. Loshchilov and F. Hutter, Decoupled weight decay regularization, in *International Conference on Learning Representations* (2019), arXiv:1711.05101.

[39] M. Raissi, P. Perdikaris, and G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. **378**, 686 (2019).

[40] S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis, Physics-informed deep neural operator networks, arXiv:2207.05748.

[41] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, Physics-informed neural operator for learning partial differential equations, arXiv:2111.03794.

[42] M. Rotman, A. Dekel, R. I. Ber, L. Wolf, and Y. Oz, Semi-supervised learning of partial differential operators and dynamical flows, arXiv:2207.14366.

[43] See the Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevD.108.L101701 for movies of the wave packet evolution in $d = 1$ and $d = 2$.