MULTISCALE MODEL. SIMUL.
Vol. 19, No. 1, pp. 344–373

MULTILEVEL FINE-TUNING: CLOSING GENERALIZATION GAPS IN APPROXIMATION OF SOLUTION MAPS UNDER A LIMITED BUDGET FOR TRAINING DATA*

ZHIHAN LI†, YUWEI FAN‡, AND LEXING YING§

Abstract. In scientific machine learning, regression networks have been recently applied to approximate solution maps (e.g., the potential-ground state map of the Schrödinger equation). In this paper, we aim to reduce the generalization error without spending more time on generating training samples. However, to reduce the generalization error, the regression network needs to be fit on a large number of training samples (e.g., a collection of potential-ground state pairs). The training samples can be produced by running numerical solvers, which takes significant time in many applications. In this paper, we aim to reduce the generalization error without spending more time on generating training samples. Inspired by few-shot learning techniques, we develop the multilevel fine-tuning algorithm by introducing levels of training: we first train the regression network on samples generated at the coarsest grid and then successively fine-tune the network on samples generated at finer grids. Within the same amount of time, numerical solvers generate more samples on coarse grids than on fine grids. We demonstrate a significant reduction of generalization error in numerical experiments on challenging problems with oscillations, discontinuities, or rough coefficients. Further analysis can be conducted in the neural tangent kernel regime, and we provide practical estimators to the generalization error. The number of training samples at different levels can be optimized for the smallest estimated generalization error under the constraint of budget for training data. The optimized distribution of budget over levels provides practical guidance with theoretical insight as in the celebrated multilevel Monte Carlo algorithm.

Key words. multilevel method, few-shot learning, generalization, parametric model, neural tangent kernel

AMS subject classifications. 65N55, 65C20, 62J07, 68Q32

DOI. 10.1137/20M1326404

1. Introduction.

1.1. Background.

1.1.1. Approximating solution maps with regression networks. Contemporary machine learning techniques, especially deep neural networks, have been recently introduced to scientific computing tasks. For problems involving partial differential equations (PDEs), neural networks can be utilized as universal approximators to the solution function in order to solve a PDE directly [39, 51, 18, 34, 54]. In this setting, to solve the PDE $\mathcal{N}u = 0$, where $\mathcal{N}$ is a differential operator and u is the solution function on domain $\Omega \subseteq \mathbb{R}^d$, one searches for a neural network NN which inputs the coordinate $\boldsymbol{x} \in \mathbb{R}^d$ and outputs an approximation to the solution function

*Received by the editors March 20, 2020; accepted for publication (in revised form) November 20, 2020; published electronically February 23, 2021.
https://doi.org/10.1137/20M1326404
Funding: The first author is partially supported by the elite undergraduate training program of the School of Mathematical Sciences at Peking University. The third author is partially supported by the National Science Foundation under award DMS-1818449 and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.
†School of Mathematical Sciences, Peking University, Beijing 100871, China (zhli16@pku.edu.cn).
‡Department of Mathematics, Stanford University, Stanford, CA 94305 USA (ywfan1989@gmail.com).
§Department of Mathematics and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305 USA (lexing@stanford.edu).

$\mathsf{NN}(\boldsymbol{x}) \approx u(\boldsymbol{x})$. Another setting tackles the problem of parametric PDE $\mathcal{N}_v u_v = 0$ with variable parameter v (e.g., coefficients, initial values, source terms). In this setting, one may train a neural network to approximate the solution map $v \mapsto u_v$ (the parameter-solution map) in order to apply it to various parameters [35]. For example, we consider the one-dimensional nonlinear Schrödinger equation on $\Omega = [0,1]$ with a periodic boundary condition and a fixed dispersion coefficient $\beta > 0$ [8, 20]:

$$-\Delta u(x) + v(x)\,u(x) + \beta u^3(x) = E u(x), \quad x \in [0,1], \tag{1.1}$$

$$\int_{[0,1]} u^2(x)\,\mathrm{d}x = 1, \quad \int_{[0,1]} u(x)\,\mathrm{d}x > 0. \tag{1.2}$$

We are interested in the solution map from potential v (parameter) to ground state u_v (the function $u_v = u$ satisfying the equations with the smallest energy E is the solution). Hence, we train a neural network which inputs potentials and outputs ground states to approximate the potential-ground state map $v \mapsto u_v$. Besides, in graphics, one may approximate the action of a Poisson solver, namely the solution map of $-\Delta u_v = v$ from source term v (parameter) to u_v (solution) for faster Eulerian fluid simulation [59, 37]. For inverse problems, one may approximate the regularized inverse operator which maps observations (parameter) to reconstructions (solution) in a data-driven context [36, 7, 22, 23, 26]. Similar settings of approximating solution maps can also be seen in signal processing [58], molecular dynamics [62], model reduction [41], and operator compression [19, 20, 21].

In this paper, we focus on the setting of approximating solution maps. For discretization, one typically chooses a fine grid of interest (in contrast to the coarse grid introduced later) and discretizes the solution map as a fine-grid numerical solver. Since the fine grid can be chosen up to practical considerations, the main problem of approximating the solution map is transformed into approximating the fine-grid solver. In order to harvest a neural network which approximates the fine-grid solver, one may fit a regression network (a neural network used as a regressor) on parameter-solution pairs generated by invoking the fine-grid solver. For example, in the nonlinear Schrödinger equation problem, we independently sample a collection of potentials $\{v_m\}_{m=1}^M$ from a distribution $\mathcal{D}$ as functions on a grid with grid step $h = 1/320$. We then compute the corresponding ground states $u_m = u_{v_m}$ by running a gradient flow solver [9] on the grid. With the parameter-solution pairs $\{(v_m, u_m)\}_{m=1}^M$, we initialize and train a neural network NN to fit the training samples (v_m, u_m).

1.1.2. Tradeoff between generalization and training data generation. The error of the regression network in approximating the fine-grid solver consists of two parts: (1) the training error on (v_m, u_m) and (2) the generalization error when applying the regression network on previously unseen samples (parameters) $v \sim \mathcal{D}$. In many cases, the neural network fits exactly at training samples because of over-parameterization [11, 42, 2], and hence the generalization error is of major concern.

The procedure described above to train a regression network lies in the framework of empirical risk minimization: we fit a statistical model (regression network) in a hypothesis space (e.g., functions representable by a neural network with bounded weights) by minimizing the empirical risk (training error) on training data, and then we evaluate the model on testing data in the hope of a low population risk (testing error). The generalization gap (generalization error) is the difference between the population risk and the empirical risk [46]. In particular, we have the following theorem bounding the generalization gap using Rademacher complexity [46].

THEOREM 1.1. *Let $\mathcal{L}$ be a family of functions from a set $\mathcal{V}$ to $[0, B]$ and $\{v_m\}_{m=1}^M$ be M independent random samples drawn from a distribution $\mathcal{D}$ on $\mathcal{V}$. Then, for any positive δ, with probability at least $1-\delta$ on sampling $\{v_m\}_{m=1}^M$, it holds for every $\ell \in \mathcal{L}$ that*

$$\mathrm{E}\,\ell(v) - \hat{\mathrm{E}}\,\ell(v) \le 2\hat{\mathfrak{R}}_v(\mathcal{L}) + 3B\sqrt{\frac{\log 1/\delta}{2M}}. \tag{1.3}$$

Here $\mathcal{L}$ is the loss class (composition of statistical models and the loss function),

$$\mathrm{E}\,\ell(v) = \mathrm{E}_{v\sim\mathcal{D}}\,\ell(v), \quad \hat{\mathrm{E}}\,\ell(v) = \frac{1}{M}\sum_{m=1}^{M}\ell(v_m) \tag{1.4}$$

are the population risk and the empirical risk, respectively, and

$$\hat{\mathfrak{R}}_v(\mathcal{L}) = \frac{1}{M}\,\mathrm{E}_\sigma \sup_{\ell\in\mathcal{L}}\sum_{m=1}^{M}\sigma_m\ell(v_m) \tag{1.5}$$

is the empirical Rademacher complexity, where σ_m are independent Bernoulli-like random variables which take the value ± 1 with equal probability.

Rademacher complexity delineates the complexity of hypothesis space and relates to the generalization gap. In various cases, $\hat{\mathfrak{R}}_v(\mathcal{L})$ decays in speed $O(1/\sqrt{M})$ (e.g., for the linear class or kernel class [10, 16]). However, if M is small, the Rademacher complexity can be poorly bounded, and so can the generalization gap.

As a result, in order to accurately approximate the fine-grid solver (and the solution map), a sufficient number of training samples are needed. However, numerical solvers may suffer from heavy scaling laws in computational complexity with respect to grid step due to stability, convergence, or randomness issues. Hence, generating a large number of training samples with the fine-grid solver may take a long time. For example, in [22], it takes about 50 seconds to solve a radiative transfer equation when generating a single parameter-solution pair. As a result, the generation of the whole dataset, which consists of 10 240 samples of such pairs, takes days and turns out to be much slower and much more expensive than the training of the regression network. We can observe the tradeoff between generalization and training data generation in this example. Hence arises the question of whether it is possible to reduce the generalization error in approximating the fine-grid solver without spending more time on generating training samples. In other words, we aim to close the generalization gap under a limited budget for training data.

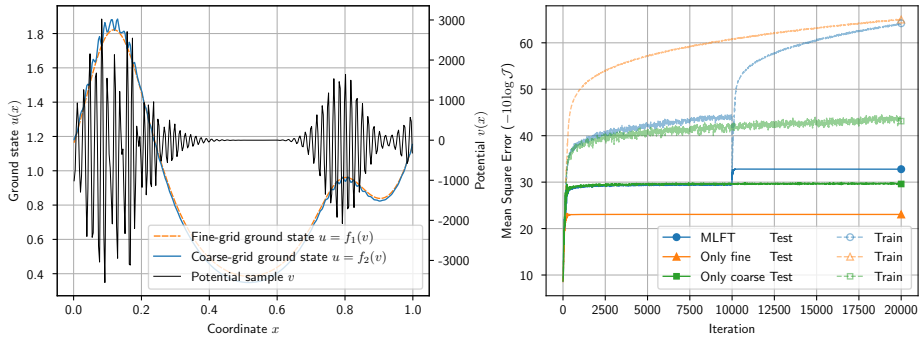
1.1.3. Multilevel fine-tuning. Few-shot learning methods have been developed to mitigate the shortage of training samples. Prior knowledge is required to either augment training samples, reduce the hypothesis space, or reach a better parameterization [60]. One particularly popular method is fine-tuning, which was initially proposed for transfer learning tasks [60]. In this approach, a model is first fit on some other tasks called source tasks with a large amount of data. The model is then refined using a similar fitting procedure on a few-shot task called a target task. The intermediate model, as a good starting point for the target task, contains some prior knowledge about the source tasks, which may be transferred to help the few-shot target task.

For image recognition tasks, models pretrained on large-scale datasets like ImageNet [56, 30] serve as excellent starting points for fine-tuning. However, fine-grid

numerical solvers to be approximated are problem-dependent. It is difficult to find a pretrained model that is well suited for multiple different problems in scientific machine learning. Meanwhile, according to multiscale methods, we may capture rough shapes of solutions on a coarse grid and then refine details on the fine grid [17]. The resemblance between fine-tuning techniques and multiscale methods motivates our multilevel fine-tuning algorithm (MLFT). Since numerical solvers on coarser grids usually occupy less time due to the scaling laws, we may generate many more training samples with a coarse-grid numerical solver. The source task of MLFT is to fit the regression network on training samples generated at the coarse grid, approximating the coarse-grid solver. Because of the large number of coarse-grid samples, we may obtain a good approximation to the coarse-grid solver. From the perspective of multiscale methods, approximation to the coarse-grid solver provides macroscopic information about the fine-grid solver (and the solution map) which we want to approximate ultimately. However, even if we have a good approximation to the coarse-grid solver, we still need to compensate for the difference between coarse-grid and fine-grid solvers. Therefore, in the target task of MLFT, we fine-tune the regression network on a few training samples generated at the fine grid, approximating the fine-grid solver. The fine-tuning step transfers information from the source task to the target task and hence reduces the generalization error in approximating the fine-grid solver. In the words of multiscale methods, microscopic information gets refined in the fine-tuning step.

In essence, the algorithm of two-level MLFT first trains the regression network on samples generated at the coarse grid and then fine-tunes the network on samples generated at the fine grid. The weights and biases learned at the previous level of the coarse grid are refined at the level of the fine grid. We take the problem of the nonlinear Schrödinger equation as an example. As mentioned before, we aim to approximate the fine-grid solver with grid step $h_2 = h = 1/320$. We assume that we generate M_2 training samples at the fine grid and each one takes time t_2. We also introduce a coarse-grid solver with grid step $h_1 = 1/40$ to generate M_1 training samples at the coarse grid, each of which takes time t_1. We observe from experiment that $t_2 = 64t_1$. The two-level MLFT algorithm consists of two stages: (1) we first initialize a regression network NN and fit it on the M_1 coarse-grid training samples and (2) we then fine-tune the regression network NN to fit the M_2 fine-grid training samples. Given the budget of time $T = 32t_2$ for generating training samples, we have the constraint $M_1t_1 + M_2t_2 = T$ or $M_1/64 + M_2 = 32$. We expect a reduction of generalization error using the two-level MLFT algorithm (e.g., $M_1 = 1024$ coarse-grid samples and $M_2 = 16$ fine-grid samples) compared to directly fitting only on $M_2 = T/t_2 = 32$ fine-grid samples. We also expect the error of MLFT to be lower than the difference between fine-grid and coarse-grid solvers, or equivalently the testing error on fine-grid samples of a regression network trained only with $M_1 = T/t_1 = 2048$ coarse-grid samples. The comparison is shown in Figure 1.

By choosing a sequence of increasingly finer grids with grid step $h_1 > h_2 > \cdots > h_L$, the two-level framework can be generalized to L levels: we first train the regression network on samples generated at the coarsest grid (h_1) and then successively fine-tune the network on samples generated at finer grids (h_2, and then h_3, and so on until h_L). We assume that M_l training samples are generated by invoking the numerical solver on the grid with grid step h_l and each one takes time t_l on average, where $t_1 < t_2 < \cdots < t_L$. If an estimator to the generalization error is available in the form $\hat{g}_L(M_1, M_2, \ldots, M_L)$, we may optimize for the smallest estimated generalization error



(a) Training sample (b) Comparison between MLFT and base-lines

FIG. 1. *The nonlinear Schrödinger equation problem. Since it is free to choose the finest grid of interest, we choose a fine grid with grid step* $h_2 = 1/320$ *and turn to approximate the fine-grid numerical solver* f_2 *with a regression network. For MLFT, we introduce a coarse grid with grid step* $h_1 = 1/40$ *and a coarse-grid numerical solver* f_1*. We plot a training sample generated by the two numerical solvers in* (a)*. In the comparison of loss curves* (b)*, setting "MLFT" trains the regression network on* $M_1 = 1024$ *coarse-grid samples and then fine-tunes on* $M_2 = 16$ *fine-grid samples, "Only coarse" trains on* $M_1 = 2048$ *coarse-grid samples, and "Only fine" trains on* $M_2 = 32$ *fine-grid samples. The networks are tested on fine-grid samples. We observe that the final testing error of MLFT is smaller than using either only coarse-grid or only fine-grid samples.*

under the constraint of budget of time T for generating training samples:

$$\begin{aligned} &\text{minimize} && \hat{g}_L\left(M_1, M_2, \ldots, M_L\right), \\ &\text{subject to} && M_1 t_1 + M_2 t_2 + \cdots + M_L t_L = T, \\ &\text{with respect to} && M_1, M_2, \ldots, M_L > 0. \end{aligned} \tag{1.6}$$

This budget distribution problem provides practical guidance for generating training samples at coarse and fine grids, which invokes the coarse-grid and fine-grid numerical solvers, respectively. In other words, it gives insight on distributing the budget for training data over levels, as in the multilevel Monte Carlo (MLMC) algorithm [25].

1.2. Related work. Structures of neural networks have inspired extensive study about hierarchical, multigrid, and multilevel methods in machine learning, especially for convolutional and residual networks [55, 24, 19, 20, 33]. In comparison, we focus on scales of grids on which numerical solvers are invoked and training samples are generated. The main goal of MLFT is to reduce the generalization error in approximating the finest-grid solver (and the solution map).

Haber et al. considered an algebraic multigrid method through the lens of optimal control [27]. With designed restriction and interpolation procedures, the neural network can be transformed between scales of grids during training. Varying the depths of networks can also be understood by temporally refining the optimal control problem. In contrast, we have coarse-grid and fine-grid training samples, while only a single neural network is used with the architecture unchanged. To match the size of training samples to the input and output shapes of the regression network, we apply spatial restriction and interpolation operators when generating training samples but not during training or fine-tuning the regression network. Besides, the problems we are considering for scientific computing suffer from scaling laws of time when generating training samples on different grids with numerical solvers. On the contrary,

classification and segmentation problems in machine learning need manual labeling, the cost of which generally does not depend on the resolution.

Very recently, Lye, Mishra, and Molinaro also noticed the tradeoff between generalization and training data generation, and they developed the multilevel machine learning Monte Carlo algorithm (ML2MC) [44]. Telescoping as in MLMC [25], they train one regression network to approximate the coarsest-grid solver and multiple networks to approximate the difference between adjacent fine-grid and coarse-grid solvers. The final model to approximate the finest-grid solver is the sum of all networks. The generalization errors also add together, similar to the variance in MLMC. In comparison, our initial motivation for MLFT follows from few-shot learning techniques. We deploy only one neural network and fine-tune between collections of samples generated at different grids. Since each level of fine-tuning reduces the distance between the current level solver and the intermediate network, generalization errors in previous levels get corrected in later levels. This results in a form of generalization error different from ML2MC. Additionally, [44] mainly considers the map from parameters to scalar observables. In this paper, we tackle the problem of approximating numerical solvers (and the solution map), the input (parameter) and output (solution) of which are functions on grids and lie in high-dimensional spaces.

Another topic related to MLFT is multifidelity modeling [52, 51, 50]. The setting of multifidelity models applies to ours: a combination of high-fidelity and low-fidelity models is accessible with different tradeoffs between efficiency and accuracy. The high-fidelity and low-fidelity models correspond to fine-grid and coarse-grid solvers, respectively. In comparison, MLFT only accesses models (invokes solvers to generate training samples) of different levels subsequently and is more coarse-grained in terms of combining high-fidelity and low-fidelity models. Meanwhile, we better leverage the nature of neural networks as parameterized statistical models, which makes fine-tuning possible. Recent progress in deep learning theory also provides us with opportunities to find practical estimators to the generalization error.

1.3. Contribution. We summarize our contribution in this paper as follows.

(1) We identify the problem of reducing generalization error of regression networks in approximation of solution maps when the budget for generating training samples is limited (section 1).

(2) We design the multilevel fine-tuning algorithm (MLFT) to reduce generalization error with inspiration from few-shot learning techniques (section 2).

(3) We perform analysis under the neural tangent kernel (NTK) regime and construct practical estimators to the generalization error, which further provides guidance to distribute the budget for training data over levels (section 3).

(4) We show the reduction of generalization error with MLFT in experiments and demonstrate optimizing the number of training samples over levels (section 4).

2. Algorithm. We proceed to present our MLFT algorithm in this section with details. As pointed out in subsection 1.1, the key idea of MLFT is to first train on samples generated at the coarsest grid and then successively fine-tune on samples generated at finer grids. We introduce the definition of levels and the procedure to generate training samples of different levels in subsection 2.1. We then explain the MLFT algorithm in subsection 2.2 together with the performance evaluation procedure of generalization error. We compare MLFT with multilevel machine learning Monte Carlo (ML2MC) [44] in subsection 2.3.

2.1. Levels and data. We tackle the problem of approximating solution maps in this paper. For simplicity, we consider problems on a d-dimensional ($d = 1, 2, 3$) domain $\Omega = [0, 1]^d$ with a periodic boundary condition. We consider the parametric PDE problem $\mathcal{N}_v u_v = 0$ where the variable parameter v lies in a function space $\mathcal{V}$. We aim to approximate the nonlinear solution map $F : \mathcal{V} \to \mathcal{U}$, which maps parameter $v \in \mathcal{V}$ to solution $u_v \in \mathcal{U}$, where $\mathcal{U}$ is the function space of solutions. In the example of the nonlinear Schrödinger equation, the solution map F is the potential-ground state map, and we can set $\mathcal{V} = \mathcal{U} = C(\Omega)$.

To numerically discretize the solution map F, we choose a finest grid of interest $\Omega_L = \{ih_L : 0 \le i < N_L\}^d$ of N_L^d evenly spaced nodes with grid step $h_L = 1/N_L$. Since we are considering regular grids, we identify functions on the grid as multidimensional arrays in $\mathbb{R}^{N_L^d}$. By discretizing the PDE $\mathcal{N}_v u_v = 0$ on grid Ω_L, we discretize the solution map F to be the finest-grid solver F_L. Since it is free to choose the finest grid step h_L according to practical considerations, the core task is to approximate the finest-grid solver F_L (discretized) instead of the solution map F (continuous). With parameters and solutions discretized as functions on grid Ω_L, we assume the spaces of discretized parameters and solutions to be $\mathcal{V}_L, \mathcal{U}_L \subseteq \mathbb{R}^{N_L^d}$, respectively. In this way, the finest-grid numerical solver on grid Ω_L is $F_L : \mathcal{V}_L \to \mathcal{U}_L$, which maps discretized parameters (functions on grid Ω_L) to discretized solutions (functions on grid Ω_L), as a discretization to the solution map $F : \mathcal{V} \to \mathcal{U}$. For example, in the one-dimensional nonlinear Schrödinger equation problem, we consider the finest-grid solver on the finest grid with grid step $h_2 = 1/320$. We may directly take the spaces of discretized potentials and ground states to be $\mathcal{V}_2 = \mathcal{U}_2 = \mathbb{R}^{320}$, and the corresponding gradient flow ground state solver [9] is represented as $F_2 : \mathcal{V}_2 \to \mathcal{U}_2$.

As explained in subsection 1.1, the limited budget for generating training samples results in large generalization error. To reduce the generalization error, MLFT introduces a series of coarser grids. The regression network is fit on training samples generated at the coarser grids before finally fine-tuning on samples generated at the finest grid Ω_L. Formally, we choose a sequence of L increasingly finer grid steps $h_1 > h_2 > \cdots > h_{L-1} > h_L$. We apply similar numerical discretization to the parametric PDE $\mathcal{N}_v u_v = 0$ on the grid $\Omega_l = \{ih_l : 0 \le i < N_l\}^d$ with $N_l = 1/h_l$ for $1 \le l \le L-1$ as in the case of the finest grid Ω_L. We assume the spaces of potentials and ground states on grid Ω_l to be $\mathcal{V}_l, \mathcal{U}_l \subseteq \mathbb{R}^{N_l^d}$, respectively, and the numerical solver working on grid Ω_l to be $F_l : \mathcal{V}_l \to \mathcal{U}_l$. For the example of the nonlinear Schrödinger equation, we use $L = 2$ levels and introduce a coarse grid with grid step $h_1 = 1/40$ for MLFT. In this case, we set $\mathcal{V}_1 = \mathcal{U}_1 = \mathbb{R}^{40}$, and the coarse-grid solver is represented as $F_1 : \mathcal{V}_1 \to \mathcal{U}_1$.

To approximate the finest-grid solver F_L, we introduce a regression network NN. By identifying multidimenisonal arrays as functions on grids, the network inputs and outputs functions on some grid. In order to capture details on the finest grid of interest Ω_L, we design the network to input and output functions on the finest grid Ω_L, as denoted by $\mathsf{NN} : \mathcal{V}_L \to \mathcal{U}_L$. In other words, the network NN works on grid Ω_L. In the example of the nonlinear Schrödinger equation, we adopt an MNN-$\mathcal{H}$ network [20] to approximate the finest-grid solver $F_2 : \mathcal{V}_2 = \mathbb{R}^{320} \to \mathcal{U}_2 = \mathbb{R}^{320}$. The input and output dimensions of the network are both $[\texttt{batch\_size}, 320]$, since the finest grid of interest Ω_2 has $N_2 = 320$ nodes.

However, for $1 \le l \le L-1$, the coarse-grid solver F_l works on the coarse grid Ω_l and is not compatible with the input and output dimensions of the network. To generate coarse-grid training samples as functions on grid Ω_L, we introduce restriction

operators $R_{L\to l} : \mathcal{V}_L \to \mathcal{V}_l$ and interpolation operators $I_{l\to L} : \mathcal{U}_l \to \mathcal{U}_L$. We transform the coarse-grid solver $F_l : \mathcal{V}_l \to \mathcal{U}_l$ into

$$f_l = I_{l\to L} \circ F_l \circ R_{L\to l} : \mathcal{V}_L \to \mathcal{U}_L, \tag{2.1}$$

which works on the finest grid Ω_L. We train the network with collections of training samples which are all functions on the finest grid Ω_L but are generated by different level-l solvers f_l. For the nonlinear Schrödinger equation problem, the coarse-grid solver $F_1 : \mathcal{V}_1 = \mathbb{R}^{40} \to \mathcal{U}_1 = \mathbb{R}^{40}$ generates potential-ground state pairs with spatial resolution $N_1 = 40$, which is incompatible with the spatial resolution of network $N_2 = 320$. Hence, we introduce a Fourier restriction operator $R_{2\to 1} : \mathcal{V}_2 = \mathbb{R}^{320} \to \mathcal{V}_1 = \mathbb{R}^{40}$ and a bicubic interpolation operator $I_{1\to 2} : \mathcal{U}_1 = \mathbb{R}^{40} \to \mathcal{U}_2 = \mathbb{R}^{320}$. The transformed function of the coarse-grid solver is $f_1 = I_{1\to 2} \circ F_1 \circ R_{2\to 1} : \mathcal{V}_2 = \mathbb{R}^{320} \to \mathcal{U}_2 = \mathbb{R}^{320}$. We use f_1 instead of F_1 to generate coarse-grid samples for training. For notational convenience, we denote $f_L = F_L$, $R_{L\to L} = \mathrm{id}_{\mathcal{V}_L}$, and $I_{L\to L} = \mathrm{id}_{\mathcal{U}_L}$.

We describe the procedure to generate samples at different levels. According to the parameters of interest, we choose a probability distribution $\mathcal{D}$ on $\mathcal{V}_L$ to sample discretized parameters. At level l where $1 \le l \le L$, we independently draw M_l parameter samples $\{v_m^l\}_{m=1}^{M_l}$ from $\mathcal{D}$ on grid Ω_L. We then restrict the parameters to grid Ω_l, invoke the coarse-grid solver F_l for solutions on grid Ω_l, and interpolate the solutions to grid Ω_L. Equivalently, we compute $u_m^l = f_l(v_m^l) = I_{l\to L} \circ F_l \circ R_{L\to l}(v_m^l)$, as depicted in Figure 2. For finding the optimized number of training samples over levels, we denote the average time to generate a sample at level l by evaluating f_l to be t_l.

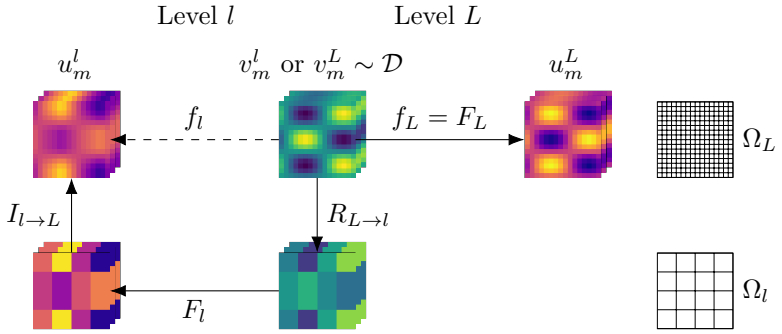


FIG. 2. *Schematic illustration on generating samples at levels l and L where $1 \le l \le L-1$. For level l, we draw parameter samples from $\mathcal{D}$ on grid Ω_L, restrict the parameters to grid Ω_l, invoke the coarse-grid solver F_l for solutions on grid Ω_l, and interpolate the solutions to grid Ω_L.*

2.2. Multilevel fine-tuning algorithm. We describe our MLFT algorithm. In order to approximate the solution map $F : \mathcal{V} \to \mathcal{U}$, we deploy a regression neural network $\mathsf{NN} : \mathcal{V}_L \to \mathcal{U}_L$ to approximate the finest-grid numerical solver $f_L : \mathcal{V}_L \to \mathcal{U}_L$. With only one level $l = L$, we fit the neural network on training samples generated at level l as described in subsection 2.1. We use the mean square error (MSE), or equivalently the squared L^2 norm on grid Ω_L, for this training (fitting) process as in Algorithm 2.1. We introduce the schematic illustration in the function space $\mathcal{V}_L \to \mathcal{U}_L$ in Figure 3.

The single-level training of a regression network at level $l = L$ is the most direct approach to approximating the finest-grid solver f_L. Single-level training at level $l = L$ serves as a baseline for comparison. As mentioned in subsection 1.1, since

Algorithm 2.1. Train a regression network at a single level l.

Generate M_l training samples $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ with $u_m^l = f_l(v_m^l)$ at level l
Initialize a neural network NN which works on grid Ω_L
Fit NN on $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ by minimizing the mean square error (MSE)

$$\mathcal{J}_l = \frac{1}{M_l} \sum_{m=1}^{M_l} \left\| u_m^l - \mathsf{NN}(v_m^l) \right\|_{L^2}^2 \tag{2.2}$$

return Trained regression neural network NN

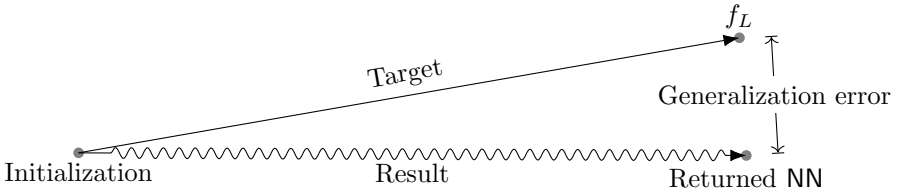


FIG. 3. *Schematic illustration of Algorithm* 2.1 *(single-level training at level $l = L$) in the function space. In this illustration, points represent functions in $\mathcal{V}_L \to \mathcal{U}_L$: We aim to approximate the finest-grid solver f_L (a function from $\mathcal{V}_L$ to $\mathcal{U}_L$) by a regression network* NN *(a parameterized function from $\mathcal{V}_L$ to $\mathcal{U}_L$). We aim to compensate for the difference between the initialized network and the finest-grid solver f_L ("Target"), and during training the network moves in the function space ("Result"). When the training error g_L^{train} vanishes, the difference between the trained regression network* NN *and the finest-grid solver f_L is the generalization error $g_L = g_L^{\text{test}}$.*

neural networks usually fit exactly at the training samples, the generalization error in approximating the finest-grid solver f_L is of major concern. Hence, we evaluate the generalization error g_L in approximating the finest-grid solver f_L for performance comparison:

$$g_L = g_L^{\text{test}} - g_L^{\text{train}}, \begin{cases} g_L^{\text{test}} = \mathrm{E}_{v\sim\mathcal{D}} \left\| f_L(v) - \mathsf{NN}(v) \right\|_2, \\ g_L^{\text{train}} = \sum_{m=1}^{M_L} \left\| u_m^L - \mathsf{NN}(v_m^L) \right\|_2 / M_L. \end{cases} \tag{2.3}$$

Here we use the vector 2-norm (instead of L^2-norm on grid Ω_L) in consistence with the notation in section 3. We expect $g_L^{\text{train}} \ll g_L^{\text{test}}$ and $g_L \approx g_L^{\text{test}}$. Another baseline for comparison is the single-level training only with samples generated at a coarse grid—for example, the single-level training at level $l = L - 1$. In this case, we evaluate the testing error for performance comparison since we do not train at level L:

$$g_L = g_L^{\text{test}} = \mathrm{E}_{v\sim\mathcal{D}} \left\| f_L(v) - \mathsf{NN}(v) \right\|_2. \tag{2.4}$$

The regression network approximates the coarse-grid solver f_{L-1} instead of the finest-grid solver f_L, so we expect $g_L \gtrsim e_L$ where e_L is the difference:

$$e_L = \mathrm{E}_{v\sim\mathcal{D}} \left\| f_L(v) - f_{L-1}(v) \right\|_2. \tag{2.5}$$

As mentioned in subsection 1.1, our MLFT algorithm first trains the regression neural network on samples generated at the coarsest grid and then successively fine-tunes the network on samples generated at finer grids. The MLFT algorithm is described in Algorithm 2.2 together with the function space illustration in Figure 4. Similar to the single-level training, MSE (2.2) is used as the loss function for the

regression network to fit the training samples. Since we deploy only one neural network NN, there is only one initialization step. In the following fine-tuning steps, we do not freeze parameters (weights and biases) of the neural network, nor do we modify the network architecture. We have not observed apparent overfitting phenomena in numerical experiments approximating solution maps and numerical solvers (e.g., Figures 1, 11 and 14), so we do not apply extra regularization either. The optimizer is not restarted, but we keep the momentum vector (for Momentum [53]) or the estimation of moments (for Adam [38]). For simplicity, parameters of the optimizer like step sizes are retained during the whole training and fine-tuning process.

Algorithm 2.2. Multilevel fine-tune (MLFT) a regression network.

Generate M_1 samples $\{(v_m^1, u_m^1)\}_{m=1}^{M_1}$ with $u_m^1 = f_1(v_m^1)$
Initialize a neural network NN which works on grid Ω_L
Fit NN on $\{(v_m^1, u_m^1)\}_{m=1}^{M_1}$ by minimizing MSE
{Train NN from initialization to f_1}
for $l = 2$ **to** L **do**
  Generate M_l samples $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ with $u_m^l = f_l(v_m^l)$
  Fit NN on $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ by minimizing MSE
  {Fine-tune NN to f_l}
end for
return Trained and fine-tuned regression neural network NN

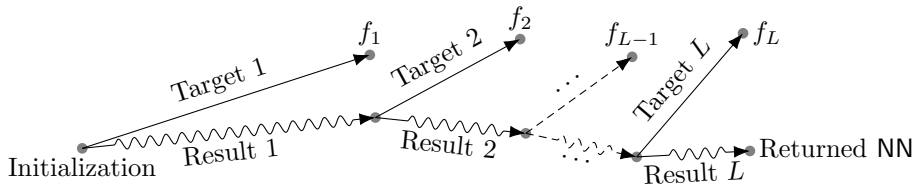


FIG. 4. *Schematic illustration of Algorithm* 2.2 *(MLFT) in the function space. We mark the training targets ("Target l") and the obtained results ("Result l") at level l, as in Figure* 3.

2.3. Multilevel machine learning Monte Carlo algorithm. In [44], Lye, Mishra, and Molinaro proposed a multilevel algorithm for regression networks called multilevel machine learning Monte Carlo (ML2MC) with inspiration from multilevel Monte Carlo (MLMC) [25]. In the paper [44], the problem of approximating parameter-observable maps of parametric PDEs is considered, where the observables are scalars. We make slight modifications to match our setting of approximating solution maps (parameter-solution maps), whose inputs and outputs are functions on grids.

The algorithm of ML2MC constructs a telescoping series as in MLMC:

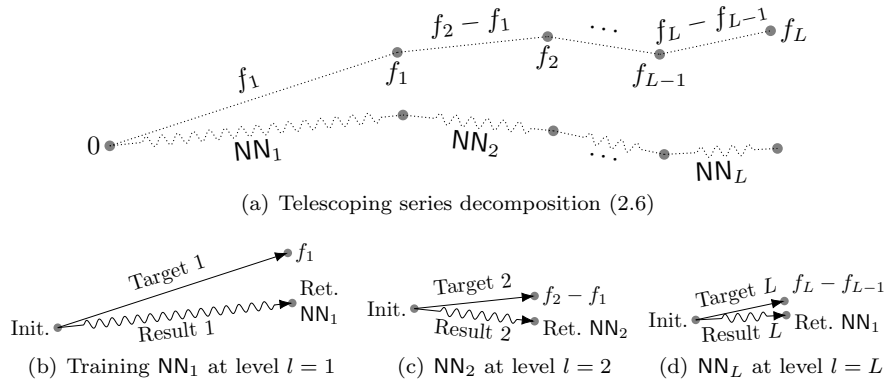
$$f_L = f_1 + (f_2 - f_1) + (f_3 - f_2) + \cdots + (f_L - f_{L-1})\,. \tag{2.6}$$

One regression network is used to approximate f_1, and the other $L-1$ networks are used to approximate $f_l - f_{l-1}$ for $2 \le l \le L$, respectively. For an approximation to the finest-grid solver f_L, one sums the L separately trained networks together, as explained in Algorithm 2.3 and Figure 5.

The main difference between MLFT (Algorithm 2.2) and ML2MC (Algorithm 2.3) is twofold. The first difference is that we fine-tune one regression network instead of

Algorithm 2.3. Multilevel machine learning Monte Carlo (ML2MC) for regression networks [44].

Generate M_1 samples $\{(v_m^1, u_m^1)\}_{m=1}^{M_1}$ with $u_m^1 = f_1(v_m^1)$
Initialize a neural network NN_1 which works on grid Ω_L
Fit NN_1 on $\{(v_m^1, u_m^1)\}_{m=1}^{M_1}$ by minimizing MSE
{Fit NN_1 from initialization to f_1}
for $l = 2$ **to** L **do**
  Generate M_l samples $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ with $u_m^l = f_l(v_m^l) - f_{l-1}(v_m^l)$
  Initialize a neural network NN_l which works on grid Ω_L
  Fit NN_l on $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ by minimizing MSE
  {Fit NN_l from initialization to $f_l - f_{l-1}$}
end for
return Sum of trained regression neural networks $\sum_{l=1}^{L} \mathsf{NN}_l$



(a) Telescoping series decomposition (2.6)

(b) Training NN_1 at level $l = 1$ (c) NN_2 at level $l = 2$ (d) NN_L at level $l = L$

FIG. 5. *Schematic illustrations of Algorithm* 2.3 *(ML2MC) in the function space. The function of the finest-grid solver f_L is decomposed by the telescoping series* (2.6) *as shown in* (a)*, and then L neural networks NN_l are trained separately at level l for $1 \le l \le L$, as shown in* (b), (c)*, and* (d).

summing several separately trained networks. We make use of the difference between parameterized model fitting and Monte Carlo estimation: fine-tuning may correct generalization errors accumulated in previous levels, while variances of multiple independent estimators (and generalization errors of several separately trained networks) add together. As a result, the form of generalization error of MLFT is different from that of ML2MC, as will be analyzed in section 3. Another difference lies in the usage of neural networks. In MLFT, we are able to confine all the computation into a single neural network thanks to the fine-tuning technique. This saves graphic memory and avoids restarting the optimizer: modern first-order optimizers memorize historical information for acceleration [53, 38]. Moreover, insights from curriculum learning [61] tell us that we may gain much more. If a parameterized statistical model is trained on easy tasks before moving on to more difficult ones, the training process on difficult tasks can get boosted significantly. In our setting, approximating the finest-grid solver f_L which contains more details can be more difficult than approximating coarse-grid solvers f_l where $1 \le l \le L-1$. Correspondingly, we observe faster convergence in the fine-tuning steps of MLFT compared to ML2MC, as shown in the experiments of section 4 and Figure 11.

3. Analysis. We analyze our algorithm of MLFT under assumptions in this section. In the MLFT algorithm, at level l, we fit the regression network on training samples $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ where $u_m^l = f_l(v_m^l)$ to approximate the solver f_l where $1 \leq l \leq L$. As a result, generalization errors are presented at the training or fine-tuning process at each level. We consider bounds of the generalization error at each level citing results of NTK and Rademacher complexity of kernel classes in subsections 3.1 and 3.2. We then chain the generalization errors at each level under guidance of empirical observations and construct a priori error estimator $\hat{g}_L$ to the generalization error g_L of MLFT in approximating the finest-grid solver in subsection 3.3. We try to get rid of pessimistic estimations and extend our estimations to finite-width cases not solidly covered by the infinite-width NTK theory by introducing practical a posteriori error estimator $\hat{g}_L$ by fitting coefficients into the form of generalization error in subsection 3.4.

In this section, we consider the training process in the function space $\mathcal{V}_L \to \mathcal{U}_L$ as in [32] and Figures 3 to 5. We denote the function of NN at initialization as $\hat{f}_0$, together with $\hat{f}_l$ for the intermediate model right after training or fine-tuning at level l. In terms of functions, the level l involves training or fine-tuning the regression network which is initially $\hat{f}_{l-1}$ to fit the target f_l on training samples ("Target l" in Figure 4), and the network eventually moves from $\hat{f}_{l-1}$ to $\hat{f}_l$ ("Result l" in Figure 4). We denote the generalization error of training or fine-tuning at level l to be

$$g_l = g_l^{\text{test}} - g_l^{\text{train}} = \mathrm{E}_{v\sim\mathcal{D}} \left\| f_l(v) - \hat{f}_l(v) \right\|_2 - \frac{1}{M_l} \sum_{m=1}^{M_l} \left\| u_m^l - \hat{f}_l(v_m^l) \right\|_2. \tag{3.1}$$

The key observation in introducing the function space illustrations is that in the NTK regime, fitting the regression neural network on training samples converges to kernel "ridgeless" regression [42]. The kernel "ridgeless" regression is linear with respect to the dependent variable (usually referred to as y in contrast to independent variables x). In our case, it turns out that at level l the increment $\hat{f}_l - \hat{f}_{l-1}$ ("Result l" in Figure 4) and the error $f_l - \hat{f}_l$ in the function space only depend on the initial difference $f_l - \hat{f}_{l-1}$ ("Target l" in Figure 4) but not on the starting point $\hat{f}_{l-1}$. This is critical to our construction of estimators to the generalization error. We briefly introduce the idea in Figure 6.

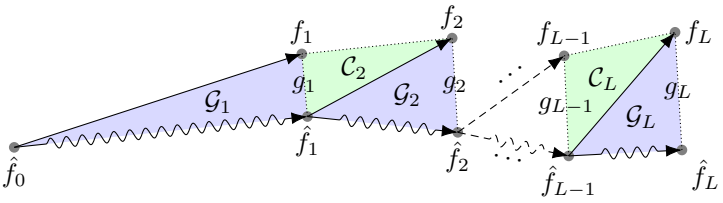


FIG. 6. *Schematic illustration of notation and the construction of error estimators (cf. Figure 4). For triangles $\mathcal{G}_l$, we estimate the generalization error g_l (i.e., $f_l - \hat{f}_l$) from the initial difference at training samples $u_m^l - \hat{f}_{l-1}(v_m^l)$ (i.e., $f_l - \hat{f}_{l-1}$). For triangles $\mathcal{C}_l$, we combine the generalization error g_{l-1} (i.e., $f_{l-1} - \hat{f}_{l-1}$) and the difference of training samples between levels $u_m^l - f_{l-1}(v_m^l)$ (i.e., $f_l - f_{l-1}$) to estimate the initial difference at training samples $u_m^l - \hat{f}_{l-1}(v_m^l)$ (i.e., $f_l - \hat{f}_{l-1}$). By chaining the triangles, we obtain an error estimator to the generalization error g_L in approximating the finest-grid solver f_L.*

3.1. Neural tangent kernel. To understand the optimization and generalization of neural networks, one particular approach is to consider the infinite-width limit

of neural networks in the function space [32, 40]. Under the NTK parameterization and random unit Gaussian initialization, channels of a neural network behave like independent samples. Hence, the law of large numbers can be cited, proving the convergence of the neural network to a Gaussian process.

Formally speaking, we denote the output of the network to be $\mathsf{NN}(v) = f(v;\theta) \in \mathbb{R}^{N_L^d}$ with input v and parameter (weights and biases) θ. We consider NTK parameterization for NN, which scales the output of each layer by $1/\sqrt{C}$, where C is the number of channels in the layer [32]. By recognizing the output $f(v;\theta)$ as a column vector, the conjugate kernel (CK), also known as the neural network Gaussian process (NNGP), is defined as

$$\Sigma(v,v') = \mathrm{E}_\theta\, f(v;\theta)\, f^{\mathsf{T}}(v';\theta) \in \mathbb{R}^{N_L^d \times N_L^d}, \tag{3.2}$$

where the expectation is taken with respect to random unit Gaussian initialization of θ. When the numbers of channels C go to infinity, Σ converges and the function $v \mapsto f(v;\theta)$ (random because of random initialization of θ) turns out to be the centered matrix-valued Gaussian process with covariance kernel Σ [32].

Similarly, the NTK is defined as

$$\Theta(v,v') = \sum_\theta \frac{\mathrm{d}f(v;\theta)}{\mathrm{d}\theta} \frac{\mathrm{d}f^{\mathsf{T}}(v';\theta)}{\mathrm{d}\theta} \in \mathbb{R}^{N_L^d \times N_L^d}, \tag{3.3}$$

where the sum is taken over all entries of the parameter θ. As pointed out in [32], Θ converges almost surely at random unit Gaussian initialization of θ in the infinite-width limit. In the case of gradient descent, NTK stays asymptotically constant during training [32]. A closed-form formula of the converged CK Σ and NTK Θ is available for various neural network architectures (including dense layers, convolutional layers, and ReLU activation layers) [5]. A software package has been developed to compute CK Σ and NTK Θ both in the infinite-width limit and for a finite-width network [48].

We summarize our assumptions for the following analysis. We assume NTK parameterization and random unit Gaussian initialization of the neural network. Moreover, we assume the network is infinitely wide so that we consider the NTK regime. The influence of finite width is discussed in subsection SM4.2. We assume that Θ is a symmetric positive definite kernel, which can be proved under assumptions (e.g., when norms of inputs are bounded both above and below, i.e., $\Theta(1)$) [32, 15]. In this case, Θ itself induces a reproducing kernel Hilbert space (RKHS), which we denote by $\mathcal{H}$. According to [14], because of the architecture of neural networks, we have $\Sigma \prec \Theta$ and hence $\hat{f}_0 \in \mathcal{H}$. Additionally, we assume that the target at each level, namely the function of the numerical solver, satisfies $f_l \in \mathcal{H}$. This means that the functions f_l are learnable under kernel "ridgeless" regression with NTK Θ. The learnability of certain functions with NTK has been proved [4].

At level l, we train a regression network to fit M_l training samples $\{(v_m^l, u_m^l)\}_{m=1}^{M}$ by minimizing MSE (equivalent to (2.2) up to a constant):

$$\mathcal{J}_l[f] = \frac{1}{M_l} \sum_{m=1}^{M_l} \left\| u_m^l - f(v_m^l;\theta) \right\|_2^2. \tag{3.4}$$

According to the framework of NTK [32], the gradient descent dynamics on (3.4) of an infinitely wide network in the function space $\mathcal{H}$ turns out to be a linear ordinary

differential equation:

$$\frac{\mathrm{d}f(v)}{\mathrm{d}t} = \sum_{\theta} \frac{\mathrm{d}f(v;\theta)}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{1}{M_l}\sum_{m=1}^{M_l} \Theta(v, v_m^l)(u_m^l - f(v_m^l)). \tag{3.5}$$

Since the training or fine-tuning process at level l fits the regression network to training samples generated by the level-l solver f_l, we have $u_m^l = f_l(v_m^l)$ and $f|_{t=0} = \hat{f}_{l-1}$. By introducing the Gram operator $\Pi \in \mathcal{L}(\mathcal{H})$, (3.5) turns out to be

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \Pi(f_l - f), \quad \Pi[f](v) = \frac{1}{M_l}\sum_{m=1}^{M_l} \Theta(v, v_m^l) f(v_m^l). \tag{3.6}$$

Since the Gram operator Π is self-adjoint, positive semidefinite, and finite-rank in $\mathcal{H}$, solution f of (3.6) converges when $t \to +\infty$. We assume that we train the regression network for a sufficiently long time at level l. In this case, the infinite-time limit $f|_{t\to+\infty} = \hat{f}_l$. Moreover, due to the assumption that NTK Θ is symmetric positive definite, the neural network indeed fits exactly at the training samples $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$. This can be summarized as the following theorem [32]. Specifically, the increment $\hat{f}_l - \hat{f}_{l-1}$ at level l ("Result l" in Figure 4) only depends on the initial difference $f_l - \hat{f}_{l-1}$ ("Target l" in Figure 4) but not on the specific function $\hat{f}_{l-1}$. Thanks to this observation, we are able to construct error estimators to the generalization error as in Figure 6 and subsection 3.3.

THEOREM 3.1. *Assume the NTK Θ to be symmetric positive definite and $\hat{f}_0, f_l \in \mathcal{H}$ for $1 \le l \le L$. Let $\mathcal{M}_l$ be the space spanned by column spaces of $\Theta(\cdot, x_m^l)$ for $1 \le m \le M_l$ and $\mathrm{P}_{\mathcal{M}_l}$ be the orthogonal projection operator onto $\mathcal{M}_l$ in $\mathcal{H}$. Then, for $1 \le l \le L$, it holds that*

$$\hat{f}_l - \hat{f}_{l-1} = \mathrm{P}_{\mathcal{M}_l}\left(f_l - \hat{f}_{l-1}\right). \tag{3.7}$$

Specifically, $\hat{f}_l \in \mathcal{H}$ for $1 \le l \le L$, and $\hat{f}_l(v_m^l) = f_l(v_m^l) = u_m^l$ for $1 \le m \le M_l$.

3.2. Generalization in the neural tangent kernel regime. We then consider the generalization error of regression network in the NTK regime (infinite-width limit). Generalization errors of neural networks have been extensively studied from the perspective of kernel methods [11, 4, 14]. Owing to the introduction of NTK, fitting a neural network is equivalent to performing kernel "ridgeless" regression with NTK Θ. As a result, the generalization error g_l in approximating the level-l solver function f_l is related to the Rademacher complexity of kernel classes with NTK Θ.

3.2.1. Kernel "ridgeless" regression. We have studied the increment $\hat{f}_l - \hat{f}_{l-1}$ at level l in the function space in Theorem 3.1. However, what we are really interested in is the RKHS norm $\left\|\hat{f}_l - \hat{f}_{l-1}\right\|_{\mathcal{H}}$, which controls the size of hypothesis space and determines the Rademacher complexity. The RKHS norm can be derived from the fact that fitting a neural network is equivalent to performing kernel "ridgeless" regression with NTK Θ [32]. (For a detailed exposition on kernel methods, see [29, 46].)

The kernel "ridgeless" regression is the limit of kernel ridge regression with diminishing ridge regularization. The kernel "ridgeless" regression has been recently found to generalize [11, 42]. In our setting, the result function $\hat{f}_l = \hat{f}_{l-1} + \mathrm{P}_{\mathcal{M}_l}(f_l - \hat{f}_{l-1})$ is

exactly the solution $\hat{f}^\star$ to the optimization problem of kernel "ridgeless" regression:

$$\begin{aligned} &\text{minimize} && \big\|\hat{f}-\hat{f}_{l-1}\big\|_{\mathcal{H}}, \\ &\text{subject to} && \hat{f}(v_m^l)=f_l(v_m^l)=u_m^l, \quad 1\le m\le M_l, \\ &\text{with respect to} && \hat{f}\in\mathcal{H}. \end{aligned} \tag{3.8}$$

Denote $\mathbf{G}_l\in\mathbb{R}^{M_lN_L^d\times M_lN_L^d}$ to be the Gram matrix and $\boldsymbol{f}_{l'}(\boldsymbol{v}^l)\in\mathbb{R}^{M_lN_L^d}$ (and also $\hat{\boldsymbol{f}}_{l'}(\boldsymbol{v}^l)$) to be vectorized functions on training samples v_m^l:

$$\mathbf{G}_l:=\begin{bmatrix} \Theta(v_1^l,v_1^l) & \Theta(v_1^l,v_2^l) & \cdots & \Theta(v_1^l,v_{M_l}^l) \\ \Theta(v_2^l,v_1^l) & \Theta(v_2^l,v_2^l) & \cdots & \Theta(v_2^l,v_{M_l}^l) \\ \vdots & \vdots & \ddots & \vdots \\ \Theta(v_{M_l}^l,v_1^l) & \Theta(v_{M_l}^l,v_2^l) & \cdots & \Theta(v_{M_l}^l,v_{M_l}^l) \end{bmatrix}, \quad \boldsymbol{f}_{l'}(\boldsymbol{v}^l):=\begin{bmatrix} f_{l'}(v_1^l) \\ f_{l'}(v_2^l) \\ \vdots \\ f_{l'}(v_{M_l}^l) \end{bmatrix}. \tag{3.9}$$

Using the kernel trick, the solution $\hat{f}_l=\hat{f}^\star$ to (3.8) can be written as (cf. (3.7)) [29]

$$\hat{f}_l=\hat{f}_{l-1}+\sum_{m=1}^{M_l}\Theta(\cdot,v_m^l)\alpha_m, \quad \boldsymbol{\alpha}=\mathbf{G}_l^{-1}\big(\boldsymbol{f}_l(\boldsymbol{v}^l)-\hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\big), \tag{3.10}$$

where $\alpha_m\in\mathbb{R}^{N_L^d}$ are column vectors and $\boldsymbol{\alpha}$ is their vectorization as in (3.9). Abbreviating $\langle\boldsymbol{u},\boldsymbol{w}\rangle_{\mathbf{G}_l^{-1}}=\boldsymbol{w}^{\mathsf{T}}\mathbf{G}_l^{-1}\boldsymbol{u}$ and $\|\boldsymbol{u}\|_{\mathbf{G}_l^{-1}}=\sqrt{\langle\boldsymbol{u},\boldsymbol{u}\rangle_{\mathbf{G}_l^{-1}}}$, the RKHS norm is

$$\big\|\hat{f}_l-\hat{f}_{l-1}\big\|_{\mathcal{H}}=\|\boldsymbol{\alpha}\|_{\mathbf{G}_l}=\big\|\boldsymbol{f}_l(\boldsymbol{v}^l)-\hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\big\|_{\mathbf{G}_l^{-1}}. \tag{3.11}$$

This result is summarized as the following theorem [29].

THEOREM 3.2. *Assume the NTK Θ to be symmetric positive definite and $\hat{f}_0, f_l\in\mathcal{H}$ for $1\le l\le L$. Then, for $1\le l\le L$,*

$$\big\|\hat{f}_l-\hat{f}_{l-1}\big\|_{\mathcal{H}}=\big\|\boldsymbol{f}_l(\boldsymbol{v}^l)-\hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\big\|_{\mathbf{G}_l^{-1}}. \tag{3.12}$$

3.2.2. Rademacher complexity of kernel classes. The complexity of hypothesis space of kernel classes can be delineated by Rademacher complexity [46]. With the RKHS norm $\big\|\hat{f}_l-\hat{f}_{l-1}\big\|_{\mathcal{H}}$ given in Theorem 3.2, we are going to consider the Rademacher complexity of $\mathcal{H}$-balls centered at $\hat{f}_{l-1}\in\mathcal{H}$.

$$\mathcal{B}\big(\hat{f}_{l-1},D\big)=\big\{f\in\mathcal{H}:\big\|f-\hat{f}_{l-1}\big\|_{\mathcal{H}}\le D\big\}. \tag{3.13}$$

We follow [45, 57] to define the Rademacher complexity of vector-valued functions (in $\mathcal{H}$) and matrix-valued kernels (NTK Θ). Given samples $\{v_m^l\}_{m=1}^{M_l}\subseteq\mathcal{V}_L$ and random testing vectors $\{\nu_m\}_{m=1}^{M_l}$ in $\mathbb{R}^{N_L^d}$, we define the empirical Rademacher complexity of a vector-valued function class $\mathcal{F}$ from $\mathcal{V}_L$ to $\mathbb{R}^{N_L^d}$ to be

$$\hat{\mathfrak{R}}_{v^l,\nu}(\mathcal{F}):=\frac{1}{M_l}\,\mathrm{E}_{\sigma,\nu}\sup_{f\in\mathcal{F}}\sum_{m=1}^{M_l}\sigma_m\nu_m^{\mathsf{T}}f(v_m^l). \tag{3.14}$$

Here σ_m are independent Bernoulli-like random variables which take the value ±1 with equal probability. As a variant of the Rademacher complexity of scalar-valued kernel classes [10, 46], we have the following theorem. Detailed proof is given in section SM1.

THEOREM 3.3. *Assume* $\left\|\Theta(v_m^l, v_m^l)\right\|_2 \le R^2$ *for* $1 \le m \le M_l$. *If the normalization condition* $\|\nu_m\|_2 \equiv 1$ *and the independence condition* $\nu_m \perp \sigma_m$ *are satisfied for* $1 \le m \le M_l$, *then it holds that*

$$\hat{\mathfrak{R}}_{v^l,\nu}\big(\mathcal{B}\big(\hat{f}_{l-1}, D\big)\big) \le \frac{DR}{\sqrt{M}}. \tag{3.15}$$

We then consider the Rademacher complexity of loss classes, for adaptation to Theorem 1.1. Since we aim to approximate the level-l solver function f_l as ground-truth at level l, we define the loss class of $\mathcal{B}\big(\hat{f}_{l-1}, D\big)$ to be

$$\mathcal{L}\big(\hat{f}_{l-1}, D; f_l, B_l\big) = \big\{v \mapsto \left\|f_l(v) - f(v)\right\|_2 \wedge B_l : f \in \mathcal{B}\big(\hat{f}_{l-1}, D\big)\big\}. \tag{3.16}$$

Here B_l is a cutoff constant to bound the loss function. We imitate [46] to derive the Rademacher complexity of loss classes of vector-valued functions in the following theorem. The proof can be found in section SM1.

THEOREM 3.4. *Assume* $\left\|\Theta\big(v_m^l, v_m^l\big)\right\|_2 \le R^2$ *for* $1 \le m \le M_l$. *Then, it holds that*

$$\hat{\mathfrak{R}}_{v^l}\big(\mathcal{L}\big(\hat{f}_{l-1}, D; f_l, B_l\big)\big) \le \frac{DR}{\sqrt{M}}. \tag{3.17}$$

3.2.3. Generalization of kernel classes. We finally consider bounds of the generalization error in approximating the level-l solver function using NTK kernel classes. As an application of Theorems 1.1 and 3.4 and a variant of [4], we have the following theorem, which is proved in section SM1. Since the network fits exactly at the training samples as in Theorem 3.1, the term of the training error is absent in (3.18).

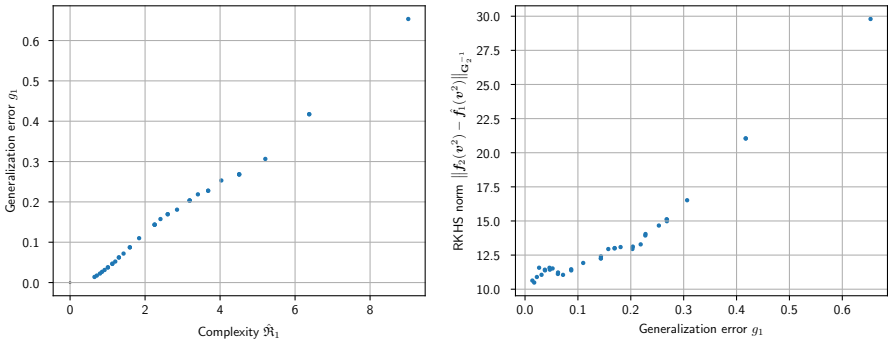
THEOREM 3.5. *Assume* $\|\Theta(v, v)\|_2 \le R^2$ *for all* v *in the support of* $\mathcal{D}$. *Then, with probability* $1 - \delta$ *on generating training samples* $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$, *it holds that*

$$\begin{aligned} &\mathrm{E}_{v\sim\mathcal{D}}\left(\left\|f_l(v) - \hat{f}_l(v)\right\|_2 \wedge B_l\right) \\ &\qquad \le \frac{2R\left\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\right\|_{\mathbf{G}_l^{-1}}}{\sqrt{M_l}} + \frac{2B_l R}{\sqrt{M_l}} + 3B_l\sqrt{\frac{\log K_l/\delta}{2M_l}}, \end{aligned} \tag{3.18}$$

where $K_l = \left\lceil \left\|f_l - \hat{f}_{l-1}\right\|_{\mathcal{H}}/B_l \right\rceil$.

3.3. A priori error estimator. We aim to find estimators to the generalization error g_L in approximating the finest-grid solver f_L. Since under assumptions the network fits exactly at the training samples at level l (cf. Theorem 3.1), the generalization error (3.1) turns out to be $g_l = \mathrm{E}_{v\sim\mathcal{D}}\left\|f_l(v) - \hat{f}_l(v)\right\|_2$. We have already mentioned the equivalence between neural network and kernel "ridgeless" regression with NTK Θ and considered the generalization error of NTK kernel classes in Theorem 3.5. The remaining task is to apply Theorem 3.5 in practical cases and chaining the generalization error as introduced in Figure 6. In this subsection, we aim to find the a priori error estimator, which can be computed before performing MLFT. We provide empirical observations to justify our construction of the estimator.

3.3.1. Estimating g_1. As mentioned in Figure 6, we apply Theorem 3.5 in the triangle $\mathcal{G}_1$ to estimate the generalization error g_1. For training at level $l = 1$, the main term on the right-hand side of (3.18) is the complexity $\hat{\mathfrak{R}}_1 = 2R\left\|\boldsymbol{f}_1(\boldsymbol{v}^1) -\right.$



(a) Between g_1 and $\hat{\mathfrak{R}}_1$ (b) Between $\left\|\boldsymbol{f}_2(\boldsymbol{v}^2) - \hat{\boldsymbol{f}}_1(\boldsymbol{v}^2)\right\|_{\mathbf{G}_2^{-1}}$ and g_1

FIG. 7. *Correlation between key quantities in the triangle $\mathcal{G}_1$ and $\mathcal{C}_2$ of Figure* 6. *Detailed discussion on the computation is presented in subsection* SM4.3.

$\hat{\boldsymbol{f}}_0(\boldsymbol{v}^1)\big\|_{\mathbf{G}_1^{-1}}/\sqrt{M_1}$. In the example of the nonlinear Schrödinger equation, we try with different M_1 and plot the correlation between complexity $\hat{\mathfrak{R}}_1$ and the generalization error g_1 in Figure 7(a).

With the clear linear correlation in Figure 7(a) as empirical justification, we consider the a priori estimator to g_1:

$$\hat{g}_1 = \frac{2Rc_1}{\sqrt{M_1}}, \quad c_1 = \left\|\boldsymbol{f}_1(\boldsymbol{v}^1) - \hat{\boldsymbol{f}}_0(\boldsymbol{v}^1)\right\|_{\mathbf{G}_1^{-1}}, \quad R = \max_{v\in\operatorname{supp}\mathcal{D}} \sqrt{\|\Theta(v,v)\|_2}. \tag{3.19}$$

For justification of (3.19) from Theorem 3.5, we need to choose an appropriate cutoff B_1 such that (1) $B_1 \gg g_1$, so that we can equate the left-hand side of (3.18) with g_1, and (2) $B_1 \ll c_1$, so that the last two terms in (3.18) are negligible. We conclude from Figure 7(a) that in practice g_1 decreases when M_1 grows (actually in speed $O(1/\sqrt{M_1})$). However, c_1 does not decrease (but actually increases as pointed out in subsection SM4.3), so such a B_1 satisfying $g_1 \ll B_1 \ll c_1$ exists for large M_1. Actually, in the example of the nonlinear Schrödinger equation, $c_1 = 56.543$ when $M_1 = 16$ (averaged over 64 collections of samples $\{(v_m^1, u_m^1)\}_{m=1}^{M_1}$) and is much greater than g_1 in Figure 7(a), so such a B_1 exists even for a small M_1. We note that the coefficients c_1 and R can be practically computed without training the regression network.

3.3.2. Estimating $\left\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\right\|_{\mathbf{G}_l^{-1}}$ for $2 \le l \le L$. As mentioned in Figure 6, in the triangle $\mathcal{C}_l$, we need to combine the generalization error g_{l-1} at the previous level and the difference between training samples $u_m^l - f_{l-1}(v_m^l) = f_l(v_m^l) - f_{l-1}(v_m^l)$ to estimate the RKHS norm $\left\|\hat{f}_l - \hat{f}_{l-1}\right\|_{\mathcal{H}} = \left\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\right\|_{\mathbf{G}_l^{-1}}$, which is critical to estimate g_l as in the case of g_1. In the example of the nonlinear Schrödinger equation, we plot the RKHS norm $\left\|\boldsymbol{f}_2(\boldsymbol{v}^2) - \hat{\boldsymbol{f}}_1(\boldsymbol{v}^2)\right\|_{\mathbf{G}_2^{-1}}$ at level $l = 2$ with different g_1 in Figure 7(b) by varying the number of training samples M_1 at level $l = 1$.

We can notice a linear correlation with positive intercept in Figure 7(b). Only when g_{l-1} is large does $\left\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\right\|_{\mathbf{G}_l^{-1}}$ deviate from the intercept. But more frequently g_{l-1} is small because we are able to generate many more samples at coarse grids, as mentioned in subsection 1.1. In this case, we may assume

$\hat{f}_{l-1} \approx f_{l-1}$ and therefore $\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}} \approx \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}} \gg \|\boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}}$. To be more precise, we make the following first-order expansion:

$$
\begin{aligned}
(3.20)\quad \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}} &= \|(\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)) + (\boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l))\|_{\mathbf{G}_l^{-1}} \\
&\approx \sqrt{\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}}^2 + 2\langle \boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l), \boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\rangle_{\mathbf{G}_l^{-1}}} \\
&\approx \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}} + \frac{\langle \boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l), \boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\rangle_{\mathbf{G}_l^{-1}}}{\|\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}}},
\end{aligned}
$$

where

$$
\begin{aligned}
(3.21)\quad &\langle \boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l), \boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\rangle_{\mathbf{G}_l^{-1}} \\
&\quad \le \left(\frac{1}{M_l}\|\boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{2,1}\right)\left(M_l\|\mathbf{G}_l^{-1}(\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l))\|_{2,\infty}\right).
\end{aligned}
$$

Here the $(2,p)$-norm takes the 2-norm on the spatial axis before taking the p-norm over the index m of training samples. We note that

$$
(3.22)\qquad \frac{1}{M_l}\|\boldsymbol{f}_{l-1}(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{2,1} = \frac{1}{M_l}\sum_{m=1}^{M_l}\|f_{l-1}(v_m^l) - \hat{f}_{l-1}(v_m^l)\|_2
$$

is a Monte Carlo estimation of the generalization error g_{l-1} due to the independence of samples $\{(v_m^{l-1}, u_m^{l-1})\}_{m=1}^{M_{l-1}}$ at level $l-1$ and $\{(v_m^l, u_m^l)\}_{m=1}^{M_l}$ at level l. In this way, we justify the estimation

$$
(3.23)\qquad \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}} \lesssim c_l + \frac{d_l g_{l-1}}{c_l},
$$

$$
(3.24)\qquad c_l = \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}}, \quad d_l = M_l\|\mathbf{G}_l^{-1}(\boldsymbol{f}_l(\boldsymbol{v}^l) - \boldsymbol{f}_{l-1}(\boldsymbol{v}^l))\|_{2,\infty}.
$$

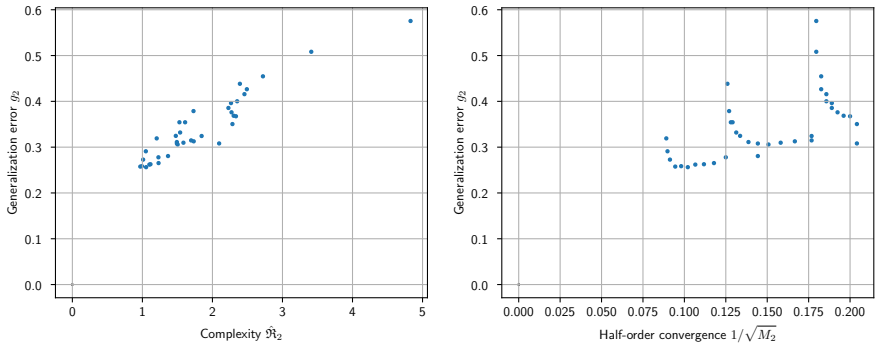
We note that the coefficients c_l and d_l can be computed without training the network. We further study their growth with respect to M_l in subsection SM4.3 empirically.

3.3.3. Estimating g_l for $2 \le l \le L$. As mentioned in Figure 6, we apply Theorem 3.5 in the triangle $\mathcal{G}_l$ to estimate the generalization error g_l from the previously discussed RKHS norm $\|\hat{f}_l - \hat{f}_{l-1}\|_{\mathcal{H}} = \|\boldsymbol{f}_l(\boldsymbol{v}^l) - \hat{\boldsymbol{f}}_{l-1}(\boldsymbol{v}^l)\|_{\mathbf{G}_l^{-1}}$. We display the correlation between the complexity $\hat{\mathfrak{R}}_2 = 2R\|\boldsymbol{f}_2(\boldsymbol{v}^2) - \hat{\boldsymbol{f}}_1(\boldsymbol{v}^2)\|_{\mathbf{G}_2^{-1}}/\sqrt{M_2}$ and the generalization error g_2 at level $l = 2$ in the example of the nonlinear Schrödinger equation in Figure 8.

We can still witness the linear correlation in Figure 8(a) during fine-tuning (cf. Figure 7(a)). Similar to the case of g_1, using the estimation to the RKHS norm (3.23), we consider the a priori estimator to g_l with coefficients (3.24):

$$
(3.25)\qquad \hat{g}_l = \frac{2R}{\sqrt{M_l}}\left(c_l + \frac{d_l \hat{g}_{l-1}}{c_l}\right).
$$

For the nonlinear Schrödinger equation problem, we consider MLFT with $L = 2$ levels as mentioned in subsection 1.1. The a priori estimators to the generalization



(a) Between g_2 and $\hat{\mathfrak{R}}_2$ (b) Between g_2 and $1/\sqrt{M_2}$

FIG. 8. *Correlation between the generalization error g_2 and the complexity $\hat{\mathfrak{R}}_2$ in the triangle $\mathcal{G}_2$ of Figure* 6. *Detailed discussion on the computation is presented in section* SM4. *We note the importance of the RKHS norm $\|\boldsymbol{f}_2(\boldsymbol{v}^2)-\hat{\boldsymbol{f}}_1(\boldsymbol{v}^2)\|_{\mathbf{G}_2^{-1}}$: If we leave it out in $\hat{\mathfrak{R}}_2$ and turn to consider $1/\sqrt{M_2}$, we observe less correlation in* (b) *compared to* (a).

error in approximating the coarse-grid and fine-grid solvers at level $l = 1$ and $l = 2$ are, respectively,

$$\hat{g}_1 = \frac{2Rc_1}{\sqrt{M_1}}, \quad \hat{g}_2 = \frac{2R}{\sqrt{M_2}}\left(c_2 + \frac{2d_2Rc_1}{c_2\sqrt{M_1}}\right), \tag{3.26}$$

where c_1, R are computed from (3.19) and c_2, d_2 computed from (3.24).

3.4. A posteriori error estimator. It is well known that a priori error estimation may be pessimistic, especially in estimating the generalization of neural networks [47, 6]. Moreover, poor estimation may give poor solutions to the budget distribution problem (1.6). Additionally, validity of the infinite-width NTK theory is questioned in finite-width scenarios, which are mostly the case in practice [1, 28]. Hence, we may try to build a practical a posteriori error estimator to the generalization error g_L in approximating the finest-grid solver f_L. Combining theoretical insights and empirical observations, the a posteriori error estimator may provide useful information about the generalization error for the budget distribution problem (1.6). The a posteriori error estimator may also extend to finite-width cases which are not solidly covered by the infinite-width NTK theory. In detail, we substitute the coefficients R, c_l, and d_l into (3.19) and (3.25) by variables a_l and b_l:

$$\hat{g}_1 = \frac{a_1}{\sqrt{M_1}}, \quad \hat{g}_l = \frac{a_l}{\sqrt{M_l}}\left(b_l + \hat{g}_{l-1}\right), \quad 2 \le l \le L. \tag{3.27}$$

We find values of the variables a_l and b_l a posteriori from trials performing MLFT with different combinations of M_l. By introducing a validation set, we compute the generalization error g_l at the trials. One particular method is the least squares method: we fit a_l and b_l by minimizing the MSE between $\log g_L$ and $\log \hat{g}_L$ over trials. The least squares method needs at least L trials to determine the function $\hat{g}_L(M_1, M_2, \ldots, M_l)$. We also propose a heuristic method using only one trial of MLFT by setting $2L-1$ equations: we set $b_l = \mathrm{E}_{v\sim\mathcal{D}}\left\|f_l(v) - f_{l-1}(v)\right\|_2$ and $g_l = \hat{g}_l$. The motivation to set b_l follows from the inequality in the triangle $\mathcal{C}_l$ of Figure 6 that $\left\|f_l(v) - \hat{f}_{l-1}(v)\right\|_2 \le \left\|f_l(v) - f_{l-1}(v)\right\|_2 + \left\|f_{l-1}(v) - \hat{f}_{l-1}(v)\right\|_2$. For ML2MC [44], the

a posteriori error estimator has the form $\hat{g}_L = \sum_{l=1}^{L} a_l/\sqrt{M_l}$. The variables a_l can also be found by the heuristic method: we set L equations equating the computed and the estimated generalization error at each level.

In the case of the nonlinear Schrödinger equation problem, we have $L = 2$ and

(3.28)
$$\hat{g}_2^{\mathrm{MLFT}}(M_1, M_2) = \frac{a_2}{\sqrt{M_2}}\left(b_2 + \frac{a_1}{\sqrt{M_1}}\right), \quad \hat{g}_2^{\mathrm{ML2MC}}(M_1, M_2) = \frac{a_1}{\sqrt{M_1}} + \frac{a_2}{\sqrt{M_2}}.$$

After determining values of the variables, one may solve the budget distribution problem (1.6) to optimize the number of training samples M_l at each level. Here the optimal M_l for ML2MC can be directly found by the Cauchy–Schwarz inequality, while the problem for MLFT is a little harder. However, we note that $1/\sqrt{\prod_{l'=1}^{l} M_{l'}}$ is convex with respect to $(M_1, M_2, \ldots, M_L)$ for $1 \le l \le L$, so highly efficient solvers can be deployed to solve the problem (1.6).

4. Numerical results. We conduct numerical experiments to show the performance of MLFT and compare between algorithms. We tackle problems with oscillations, discontinuities, or rough coefficients. For these problems, the finest-grid solvers contain much detail. Hence, it poses challenges for the regression network to have a small generalization error when there is a limited budget for generating training samples.

We first implement the neural networks in PyTorch [49] and then migrate to JAX [12] and Stax [13] in order to use the package Neural Tangents [48] to calculate NTK. The codes of experiments were run on Nvidia Tesla K80. We separately generate three sets of samples for training, validation, and testing, respectively, according to the procedure described in subsection 2.1. The generalization errors to construct the a posteriori error estimator are computed on the validation set, while the reported errors (2.3) and (2.4) in the figures are computed on the testing set. The batch size of 32 is used for all the experiments. We fix the seed of random batch and random initialization for better comparison. We train the network for 10 000 iterations at each level when we observe that the training error is smaller than the testing error (e.g., in Figures 1, 11 and 14).

4.1. Nonlinear Schrödinger equation with oscillatory potentials. We first consider the example of the one-dimensional nonlinear Schrödinger equation, or, namely, the Gross–Pitaevskiii equation. This equation relates to Bose–Einstein condensation [3, 8] and is receiving emerging research attention [31]. As already introduced in subsection 1.1, we consider (1.1) and (1.2) on domain $\Omega = [0, 1]$ with periodic boundary condition. The solution map we are interested in is the potential-ground state map.

We set the dispersion coefficient to be $\beta = 100$. The distribution $\mathcal{D}$ of potential v is generated by

(4.1)
$$v(x) = \sum_{k=1}^{K} A_k\left(\alpha + \cos\left(2\pi\omega_k x + \phi_k\right)\right)\frac{1}{\sqrt{2\pi}\sigma_k}\exp\left(-\frac{d(x, c_k)^2}{2\sigma_k^2}\right), \quad x \in [0, 1],$$

where $d(x, y) = x - y - \lfloor x - y + 1/2 \rfloor \in [-1/2, 1/2)$. We set $K = 4$, $\alpha = 0.1$ and sample $A_k \sim \mathcal{U}[-400, -200]$, $\omega_k \sim \mathcal{U}[40, 80]$, $1/\sigma_k \sim \mathcal{U}[10, 20]$, and $\phi_k \sim \mathcal{U}[0, 2\pi]$ independently. The ground state u is calculated by the gradient flow solver described in [9] with time step $\tau = 1$.

As mentioned in subsection 1.1, we consider $L = 2$ levels with grid steps $h_1 = 1/N_1 = 1/40$ and $h_2 = 1/N_2 = 1/320$, respectively. From the visualization of potential-ground state pair in Figure 1, we notice that large high-frequency components of the potential v produce small oscillations in the ground state u. Hence, we select $R_{2\to1} : \mathcal{V}_2 \to \mathcal{V}_1$ to be a Fourier restriction operator since the frequency of oscillations in the potential v exceeds the Nyquist frequency of the coarse grid. On the coarse grid only low-frequency components of the potential v remain, and we select $I_{1\to2} : \mathcal{U}_1 \to \mathcal{U}_2$ to be the cubic interpolation operator. We observe that running times of coarse-grid and fine-grid solvers approximately satisfy $t_2 = 64t_1$, as discussed in subsection SM3.1.

We use the MNN-$\mathcal{H}$ network [19] as the regression network. We use a network with 6 branches where each branch is a 5-layer 160-channel subnetwork as specified in section SM2. We use Momentum [53] as the optimizer with momentum coefficient 0.975 and learning rate 10. The theory of NTK is proved to be applicable for the Momentum optimizer [40]. The learning rate is seemingly huge but is applicable in the NTK parameterization [32].

Recall that M_1 and M_2 are the numbers of training samples generated by the coarse-grid and fine-grid solvers f_1 and f_2, respectively. We aim to reduce the generalization error g_2 under a fixed budget for generating training samples, i.e., $M_1t_1 + M_2t_2 = T$. To explore different combinations of M_1 and M_2 under a fixed budget T, we define the coarse-to-total ratio $r = M_1t_1/(M_1t_1 + M_2t_2)$ to be the proportion of budget spent in generating coarse-grid samples. We examine the relationship between the error g_2 ((2.3) and (2.4)) and the coarse-to-total ratio r under a budget fixed T in Figure 9. We also plot the error estimators $\hat{g}_2$ to the generalization error introduced in subsections 3.3 and 3.4. The heuristic method to construct the a posteriori error estimator runs a trial of MLFT with $M_1 = 64$ coarse-grid samples and $M_2 = 32$ fine-grid samples.

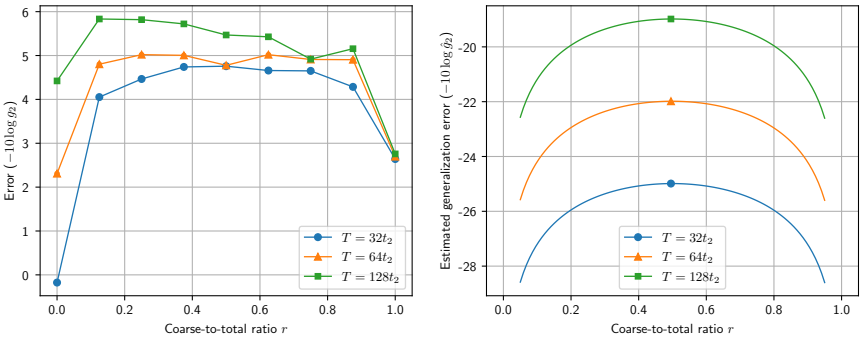
From the experiment, we observe that MLFT provides a significant reduction of the generalization error in approximating the fine-grid solver with a regression network. The margin is larger when we are given a smaller budget for generating training samples. We notice that the r with the smallest estimated generalization error decreases when the budget grows. The a posteriori error estimators correctly capture this tendency of decreasing r.

4.2. Viscous Burgers equation with discontinuous initial values. The Burgers equation plays an essential role in fluid dynamics and traffic flow modeling. Since the viscous Burgers equation contains both diffusion and nonlinear convection terms, it is frequently taken as an example for conservation law solvers and model reduction methods [41]. The one-dimensional viscous Burgers equation for $u(x;t)$ is

$$u_t + \left(\frac{u^2}{2}\right)_x = \kappa u_{xx}, \quad x \in \Omega. \tag{4.2}$$

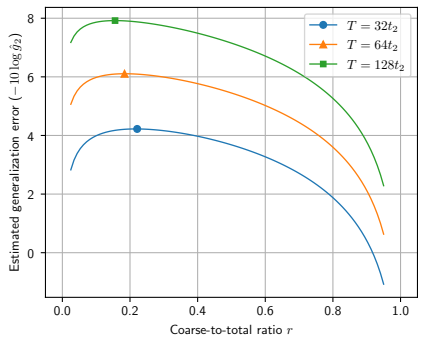
We consider the equation on $\Omega = [0,1]$ with a periodic boundary condition. The solution map we are approximating is the time evolution map. Formally, we are interested in the solution map $v \mapsto u_v$, which takes initial value $v = u(\cdot;0)$ as parameter and gives terminal value $u_v = u(\cdot;t_{\text{term}})$ as solution.

We consider the problem with discontinuous initial values. In detail, the distri-

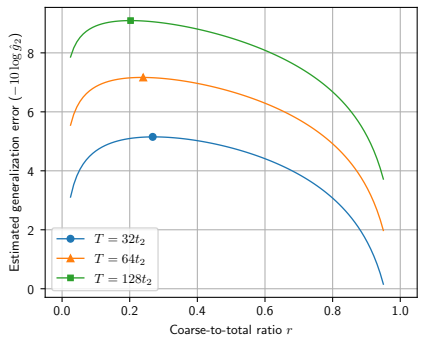


(a) Experiment

(b) A priori



(c) A posteriori, least square



(d) A posteriori, heuristic

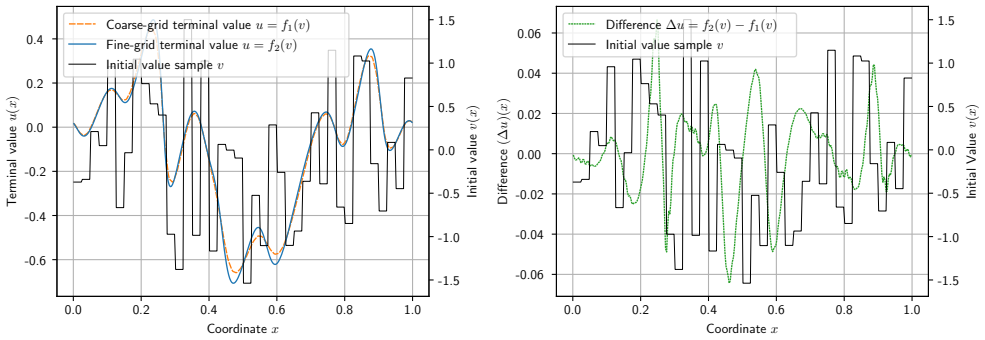
FIG. 9. *Error g_2 (2.3) and (2.4) and the estimated generalization error $\hat{g}_2$ for the nonlinear Schrödinger equation problem. In experiment* (a) *when $r = 0$ we use only fine-grid samples to train the regression network (i.e., single-level $l = 2$), and when $r = 1$ we use only coarse-grid samples (i.e., single-level $l = 1$). We observe that MLFT with a selected r is able to outperform single-level training with either $l = 1$ or 2. In particular, the combination of $T = 32t_2$ and $r = 1/2$ has smaller generalization error than $T = 128t_2$ and $r = 0$. This means that MLFT achieves the same generalization error while saving 75% of the budget for generating training samples. For error estimators, we observe that the a priori error estimator* (b) *is pessimistic. The a posteriori error estimators capture the generalization error better. The r with the smallest estimated generalization error is marked as the solution to the budget distribution problem* (1.6). *A significant reduction of generalization error is still presented in experiments with these r optimized from a posteriori error estimators.*

bution $\mathcal{D}$ of parameter v consists of step functions

$$v(x) = \sum_{k=1}^{K} A_k \chi_{[(k-1)/K, k/K)}(x), \quad x \in [0,1]. \tag{4.3}$$

We set K to be 40 and sample $A_k \sim \mathcal{N}(0,1)$ independently for $1 \le k \le K$. We consider the solution at terminal time $t_{\text{term}} = 0.1$ under diffusivity coefficient $\kappa = 0.005$. For the numerical solver F_l at each level, we use the first-order operator splitting scheme which includes the Godunov scheme for convection and the explicit Euler scheme for diffusion. For stability reasons, we choose the time step to be $10h^2$ on the grid with grid step h. As a result, the time to run the level-l solver satisfies $t_l \propto 1/h_l^3$. We choose the average operator to be $R_{L\to l} : \mathcal{V}_L \to \mathcal{V}_l$ and the linear interpolation operator to be $I_{l\to L} : \mathcal{U}_l \to \mathcal{U}_L$.

We consider MLFT with $L = 2$ levels with grid steps $h_1 = 1/N_1 = 1/80$ and $h_2 = 1/N_2 = 1/320$. In this case, $t_2 = 64t_1$. We visualize a training sample, namely an initial value–terminal value pair in Figure 10. We use a 9-layer convolutional neural network with 160 channels as the regression network, as specified in section SM2. We use the Momentum optimizer with learning rate 10 and momentum coefficient 0.975. In this problem, MLFT still reduces the generalization error in approximating the fine-grid solver as shown in Figure 11. We compute the error g_2 (2.3) and (2.4) of MLFT and ML2MC with different coarse-to-total ratio r under a fixed budget T. We also plot the estimated generalization error $\hat{g}_2$ of the a posteriori estimator constructed by the heuristic method which runs a trial of MLFT with $M_1 = 64$ coarse-grid samples and $M_2 = 32$ fine-grid samples.



(a) Training sample (b) Difference

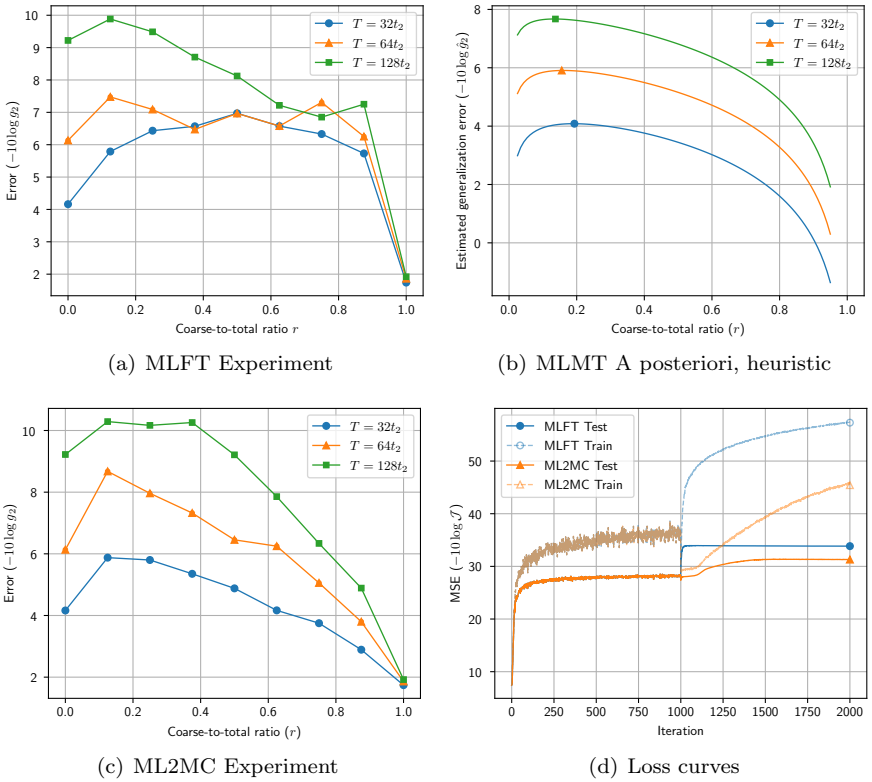
FIG. 10. *Visualization of a training sample of the viscous Burgers equation problem. The coarse-grid sample suffers from numerical dissipation as shown in* (a)*, which causes the difference between coarse-grid and fine-grid samples in* (b).

From the experiments, we find that both MLFT and ML2MC reduce the generalization error g_2 in approximating the finest-grid solver f_2. In the case that the budget for generating training samples is small, MLFT reduces the generalization error slightly better than ML2MC. Besides, we find that at the fine level $l = 2$ ML2MC converges much slower than MLFT. This is because of the effect of curriculum learning [61], as mentioned in Algorithm 2.3. In MLFT, the convergence of fine-tuning steps is boosted by the previous training or fine-tuning steps. However, in ML2MC, networks are trained separately and converge separately.

4.3. Diagonal of inverse to elliptic operators with rough coefficients. The inverse to an elliptic operator is known as the Green function, which serves as the fundamental solution to elliptic PDE. As an example, we are particularly interested in the diagonal of inverse $u(\boldsymbol{x}) = g(\boldsymbol{x}, \boldsymbol{x})$ to the Schrödinger operator, namely $g = (-\Delta + v)^{-1}$. This problem has been studied in [43, 21] and has applications in quantum chemistry. We aim to approximate the potential-diagonal of inverse map of the Schrödinger operator, which takes potential $v(\boldsymbol{x})$ as parameter and gives the diagonal of inverse $u_v(\boldsymbol{x}) = u(\boldsymbol{x})$ to the corresponding Schrödinger operator.

Let us tackle the two-dimensional problem with $\boldsymbol{x} = (x, y)$ on $\Omega = [0, 1]^2$ with a periodic boundary condition. We generate rough potentials v by

$$v(x, y) = \sum_{k=1}^{K} A_k \cos\left(2^k \pi (x + y) + \phi_k\right) + \sum_{k=1}^{K} B_k \cos\left(2^k \pi (2x - y) + \psi_k\right) + C. \tag{4.4}$$



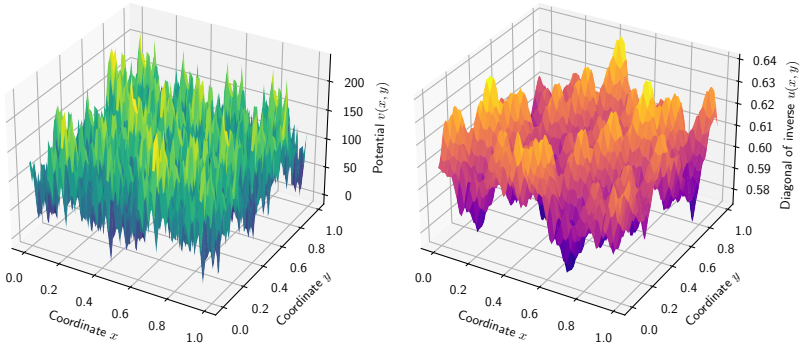
(a) MLFT Experiment (b) MLMT A posteriori, heuristic

(c) ML2MC Experiment (d) Loss curves

FIG. 11. *Error g_2 (2.3) and (2.4) and the estimated generalization error $\hat{g}_2$ for the viscous Burgers equation problem. We observe that in experiments* (a) *the combination of $T = 32t_2$ and $r = 1/2$ with MLFT has smaller generalization error than $T = 64t_2$ and $r = 0$, achieving a 50% saving of the budget. The a posteriori error estimator* (b) *captures the trend of error curves in* (a). *In comparison between MLFT and ML2MC in* (c), *we observe that MLFT reduces the generalization error slightly better than ML2MC in the case $T = 32t_2$, namely when the budget is small. We plot the loss curves of MLFT and ML2MC in the case $M_1 = 1024$ and $M_2 = 16$ in* (d) *and find that at level $l = 2$ MLFT converges much faster (in less than 500 iterations) than ML2MC (in about 5000 iterations).*

For the distribution $\mathcal{D}$ of rough potentials, we set $K = 6$, $C = 100$ and sample $A_k, B_k \sim \mathcal{N}(0, 20^2)$ and $\phi_k, \psi_k \sim \mathcal{U}[0, 2\pi]$ independently. We visualize a potential-diagonal of inverse pair in Figure 12. We invoke the SelInv algorithm [43] for the numerical solvers F_l, which takes $O(1/h^3)$ times on a two-dimensional grid with grid step $h = 1/N$. We choose $R_{L\to l} : \mathcal{V}_L \to \mathcal{V}_l$ to be the Fourier restriction operator and $I_{l\to L} : \mathcal{U}_l \to \mathcal{U}_L$ to be the cubic interpolation operator.

We consider four different grids with grid step $h_1 = 1/N_1 = 1/10$, $h_2 = 1/N_2 = 1/20$, $h_3 = 1/N_3 = 1/40$, and $h_4 = 1/N_4 = 1/80$, respectively. In this case, $t_4 = 8t_3 = 64t_2 = 512t_1$. We aim to approximate the finest-grid solver f_4 on the finest grid with grid step $h_4 = 1/80$. We use a two-dimensional MNN-$\mathcal{H}$ network with 4 branches and 5 layers as the regression network, as specified in section SM2. We use the Adam optimizer with learning rate 3.0×10^{-4} [38]. Although the theory of NTK is not applicable in this case, we still consider the ability of MLFT in reducing the generalization error. We perform MLFT with different numbers of levels L for $1 \le L \le 4$: we successively use training samples generated at the grid with grid step h_{4+l-L}



(a) Potential sample v (b) Diagonal of inverse u

FIG. 12. *Visualization of a training sample of the diagonal of inverse problem. We observe high-frequency oscillations in the potential* v (a)*, but the diagonal of the inverse* u *to the corresponding Schrödinger operator* (b) *is smoother.*

for $l = 1, 2, \ldots, L$. The number of training samples M_l at different levels is optimized according to the budget distribution problem (1.6). Here we use the a posteriori error estimator constructed by the heuristic method which runs trials with $M_1 = 256$, $M_2 = 128$, $M_3 = 64$, and $M_4 = 32$. We compute and report the testing error g_4^{test} with different numbers of levels L under a fixed budget T in Figure 13. We observe that as the number of levels grows, MLFT achieves smaller error in approximating the finest-grid solver under a fixed budget for generating training samples.

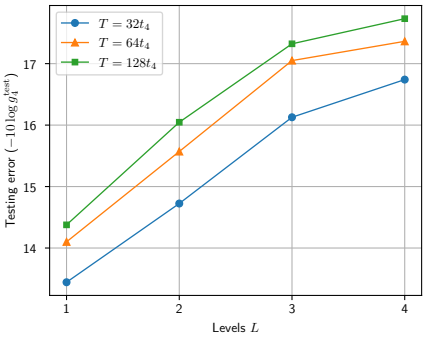
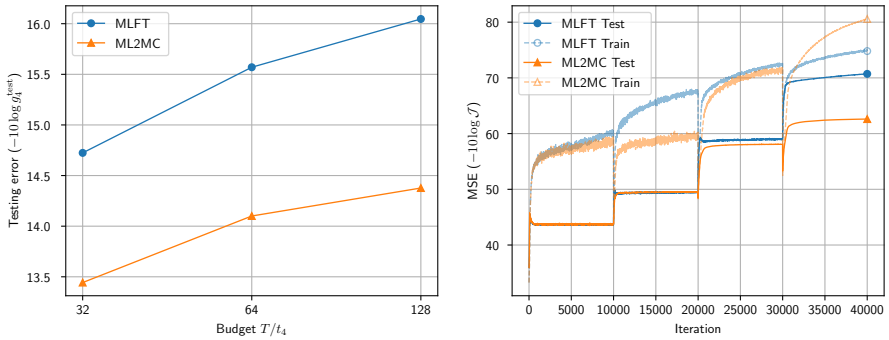


FIG. 13. *Testing error* g_4^{test} *for the diagonal of inverse problem. We observe that the error is significantly reduced under a fixed budget* T *when the number of levels increases.*

We compare MLFT and ML2MC with $L = 4$ levels under different budgets T for generating training samples in Figure 14. We compute the testing error g_4^{test} to compare the accuracy of MLFT and ML2MC. We also plot the loss curves in order to understand the convergence and compare the efficiency. For reference, we show the optimized number of training samples at different levels in Table 1 as the solution to the budget distribution problem (1.6).

5. Conclusion. We establish the multilevel fine-tuning algorithm (MLFT), Algorithm 2.2, in this paper. We aim to reduce the generalization error of regression networks in approximating solution maps and numerical solvers without spending



(a) Testing error g_4^{test} (b) Loss curves

FIG. 14. *Comparison between MLFT and ML2MC for the diagonal of inverse problem. In terms of accuracy* (a)*, we observe that MLFT reduces the generalization error better than ML2MC. In terms of convergence* (b)*, we observe that MLFT converges faster than ML2MC at the level* $l = 3$*. We note that the difference between MLFT and ML2MC becomes larger after entering level* $l = 4$*. This is because MLFT corrects the generalization error in fine-tuning steps while ML2MC accumulates the generalization error. The loss curves are extracted from the experiments in* (a) *under the fixed budget* $T = 32t_4$*.*

TABLE 1

Optimized number of training samples at different levels for the diagonal of inverse problem. The number of training samples is found by solving the budget distribution problem (1.6) *and is subject to* $M_1t_1 + M_2t_2 + M_3t_3 + M_4t_4 = T$ *or* $M_1/512 + M_2/64 + M_3/8 + M_4 = T/t_4$*. We observe that compared to MLFT, ML2MC generates many more coarse-grid samples in exchange for some fine-grid samples.*

	T/t_4	M_1	M_2	M_3	M_4
MLFT Algorithm 2.2	32	53	51	46	25
	64	68	72	76	53
	128	86	101	126	111
ML2MC Algorithm 2.3	32	2143	653	54	7
	64	4286	1305	107	13
	128	8572	2611	214	27

more time on generating training samples. Inspired by the resemblance between fine-tuning techniques and multiscale methods, we train the regression networks on samples generated at the coarse grid and fine-tune on samples generated at finer grids. In experiments, we achieve significant reduction of the generalization error with MLFT under a fixed budget for generating training samples, compared with baselines trained with only coarse-grid or only fine-grid samples. Thanks to the neural tangent kernel (NTK) theory, we are able to perform analysis on the MLFT algorithm under assumptions. Although it is notoriously difficult to find sharp bounds on the generalization error of neural networks, we make use of the form of generalization error provided by statistical machine learning theory and find the coefficients a posteriori. By minimizing the estimated generalization error given by the practical a posteriori error estimator, we distribute the budget for generating training samples over levels. In this way, we hope to provide practical guidance for reducing the generalization error with theoretical insight.

We have not exhaustively tested MLFT on many other intriguing numerical problems in scientific machine learning. Problems with heavier scaling of time with respect to the grid step may enjoy greater reduction of generalization error with the help of MLFT. It is also possible to generalize MLFT to other settings besides regression networks. For the error estimators, we are interested in other methods to construct the a posteriori error estimators besides the two mentioned in this paper. More detailed analysis can be done, possibly providing new forms of the generalization error.

Acknowledgments. We would like to thank Tianle Cai for providing references. We also appreciate the discussion with Professor Bin Dong on the delivery of the subject. We would like to express our gratitude to the anonymous reviewers together with Yiqing Xie, Weijie Chen, Jikai Hou, and Zeyu Zhao for careful proofreading and kind comments on the manuscript, which greatly improved the quality of the paper.

REFERENCES

[1] Z. Allen-Zhu, Y. Li, and Y. Liang, *Learning and generalization in overparameterized neural networks, going beyond two layers*, in Advances in Neural Information Processing Systems 32 (Vancouver, Canada), Curran Associates, Red Hook, NY, 2019, pp. 6158–6169.

[2] Z. Allen-Zhu, Y. Li, and Z. Song, *A convergence theory for deep learning via over-parameterization*, in Proceedings of the 36th International Conference on Machine Learning (Long Beach, CA), PMLR, 2019, pp. 242–252.

[3] J. R. Anglin and W. Ketterle, *Bose–Einstein condensation of atomic gases*, Nature, 416 (2002), pp. 211–218, https://doi.org/10.1038/416211a.

[4] S. Arora, S. Du, W. Hu, Z. Li, and R. Wang, *Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks*, in Proceedings of the 36th International Conference on Machine Learning (Long Beach, CA), PMLR, 2019, pp. 322–332.

[5] S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang, *On exact computation with an infinitely wide neural net*, in Advances in Neural Information Processing Systems 32 (Vancouver, Canada), Curran Associates, Red Hook, NY, 2019, pp. 8141–8150.

[6] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang, *Stronger generalization bounds for deep nets via a compression approach*, in Proceedings of the 35th International Conference on Machine Learning (Stockholm, Sweden), PMLR, 2018, pp. 254–263.

[7] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, *Solving inverse problems using data-driven models*, Acta Numer., 28 (2019), pp. 1–174, https://doi.org/10.1017/S0962492919000059.

[8] V. S. Bagnato, D. J. Frantzeskakis, P. G. Kevrekidis, B. A. Malomed, and D. Mihalache, *Bose–Einstein condensation: Twenty years after*, Rom. Rep. Phys., 67 (2015), pp. 5–50.

[9] W. Bao and Q. Du, *Computing the ground state solution of Bose–Einstein condensates by a normalized gradient flow*, SIAM J. Sci. Comput., 25 (2004), pp. 1674–1697, https://doi.org/10.1137/S1064827503422956.

[10] P. L. Bartlett and S. Mendelson, *Rademacher and Gaussian complexities: Risk bounds and structural results*, in Computational Learning Theory, COLT 2001, Lecture Notes in Comput. Sci. 2111, Springer, Berlin, Heidelberg, 2001, pp. 224–240, https://doi.org/10.1007/3-540-44581-1_15.

[11] M. Belkin, S. Ma, and S. Mandal, *To understand deep learning we need to understand kernel learning*, in Proceedings of the 35th International Conference on Machine Learning (Stockholm, Sweden), PMLR, 2018, pp. 541–549.

[12] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: Composable Transformations of Python+NumPy Programs*, http://github.com/google/jax, 2018.

[13] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *Stax, a Flexible Neural Net Specification Library in JAX*, https://github.com/google/jax/blob/master/jax/experimental/stax.py, 2018.

[14] Y. Cao and Q. Gu, *Generalization bounds of stochastic gradient descent for wide and deep neural networks*, in Advances in Neural Information Processing Systems 32 (Vancouver,

Canada), Curran Associates, Red Hook, NY, 2019, pp. 10836–10846.
[15] Y. Cao and Q. Gu, *Generalization error bounds of gradient descent for learning over-parameterized deep ReLU networks*, Proc. AAAI Conf. Artif. Intell., 34, (2020), pp. 3349–3356, https://doi.org/10.1609/aaai.v34i04.5736.
[16] C. Cortes, M. Kloft, and M. Mohri, *Learning kernels using local Rademacher complexity*, in Advances in Neural Information Processing Systems 26 (Lake Tahoe, NV), Curran Associates, Red Hook, NY, 2013, pp. 2760–2768.
[17] W. E and B. Engquist, *The heterogenous multiscale methods*, Commun. Math. Sci., 1 (2003), pp. 87–132.
[18] W. E and B. Yu, *The Deep Ritz Method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12, https://doi.org/10.1007/s40304-018-0127-z.
[19] Y. Fan, J. Feliu-Fabà, L. Lin, L. Ying, and L. Zepeda-Núñez, *A multiscale neural network based on hierarchical nested bases*, Res. Math. Sci., 6 (2019), 21, https://doi.org/10.1007/s40687-019-0183-3.
[20] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núñez, *A multiscale neural network based on hierarchical matrices*, Multiscale Model. Simul., 17 (2019), pp. 1189–1213, https://doi.org/10.1137/18M1203602.
[21] Y. Fan, C. Orozco Bohorquez, and L. Ying, *BCR-Net: A neural network based on the nonstandard wavelet form*, J. Comput. Phys., 384 (2019), pp. 1–15, https://doi.org/10.1016/j.jcp.2019.02.002.
[22] Y. Fan and L. Ying, *Solving Optical Tomography with Deep Learning*, preprint, https://arxiv.org/abs/1910.04756, 2019.
[23] Y. Fan and L. Ying, *Solving electrical impedance tomography with deep learning*, J. Comput. Phys., 404 (2020), 109119, https://doi.org/10.1016/j.jcp.2019.109119.
[24] R. Gao, Y. Lu, J. Zhou, S.-C. Zhu, and Y. N. Wu, *Learning generative ConvNets via multi-grid modeling and sampling*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Salt Lake City, UT), IEEE, Washington, DC, 2018, pp. 9155–9164, https://doi.org/10.1109/CVPR.2018.00954.
[25] M. B. Giles, *Multilevel Monte Carlo path simulation*, Oper. Res., 56 (2008), pp. 607–617, https://doi.org/10.1287/opre.1070.0496.
[26] D. Gilton, G. Ongie, and R. Willett, *Neumann networks for linear inverse problems in imaging*, IEEE Trans. Comput. Imaging, 6 (2020), pp. 328–343, https://doi.org/10.1109/TCI.2019.2948732.
[27] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, *Learning across scales—multiscale methods for convolution neural networks*, in Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18) (New Orleans, LA), AAAI Press, 2018, pp. 3142–3148.
[28] B. Hanin and M. Nica, *Finite depth and width corrections to the neural tangent kernel*, in 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, 2020, https://openreview.net/forum?id=SJgndT4KwB.
[29] T. Hofmann, B. Schölkopf, and A. J. Smola, *Kernel methods in machine learning*, Ann. Statist., 36 (2008), pp. 1171–1220, https://doi.org/10.1214/009053607000000677.
[30] M. Huh, P. Agrawal, and A. A. Efros, *What Makes ImageNet Good for Transfer Learning?*, preprint, https://arxiv.org/abs/1608.08614, 2016.
[31] B. Ilan and M. I. Weinstein, *Band-edge solitons, nonlinear Schrödinger/Gross–Pitaevskii equations, and effective media*, Multiscale Model. Simul., 8 (2010), pp. 1055–1101, https://doi.org/10.1137/090769417.
[32] A. Jacot, F. Gabriel, and C. Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, in Advances in Neural Information Processing Systems 31 (Montreal, Canada), Curran Associates, Red Hook, NY, 2018, pp. 8571–8580.
[33] H. Juncai and X. Jinchao, *MgNet: A unified framework of multigrid and convolutional neural network*, Sci. China Math., 62 (2019), pp. 1331–1354, https://doi.org/10.1007/s11425-019-9547-2.
[34] Y. Khoo, J. Lu, and L. Ying, *Solving for high-dimensional committor functions using artificial neural networks*, Res. Math. Sci., 6 (2018), 1, https://doi.org/10.1007/s40687-018-0160-2.
[35] Y. Khoo, J. Lu, and L. Ying, *Solving parametric PDE problems with artificial neural networks*, European J. Appl. Math., (2020), https://doi.org/10.1017/S0956792520000182.
[36] Y. Khoo and L. Ying, *SwitchNet: A neural network model for forward and inverse scattering problems*, SIAM J. Sci. Comput., 41 (2019), pp. A3182–A3201, https://doi.org/10.1137/18M1222399.

[37] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, *Deep fluids: A generative network for parameterized fluid simulations*, Comput. Graph. Forum, 38 (2019), pp. 59–70, https://doi.org/10.1111/cgf.13619.

[38] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings, San Diego, CA, 2015; preprint, https://arxiv.org/abs/1412.6980, 2014.

[39] I. Lagaris, A. Likas, and D. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw. Learn. Syst., 9 (1998), pp. 987–1000, https://doi.org/10.1109/72.712178.

[40] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, *Wide neural networks of any depth evolve as linear models under gradient descent*, in Advances in Neural Information Processing Systems 32 (Vancouver, Canada), Curran Associates, Red Hook, NY, 2019, pp. 8572–8583.

[41] K. Lee and K. T. Carlberg, *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*, J. Comput. Phys., 404 (2020), 108973, https://doi.org/10.1016/j.jcp.2019.108973.

[42] T. Liang and A. Rakhlin, *Just interpolate: Kernel "Ridgeless" regression can generalize*, Ann. Statist., 48 (2020), pp. 1329–1347, https://doi.org/10.1214/19-AOS1849.

[43] L. Lin, J. Lu, L. Ying, R. Car, and W. E, *Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems*, Commun. Math. Sci., 7 (2009), pp. 755–777.

[44] K. O. Lye, S. Mishra, and R. Molinaro, *A multi-level procedure for enhancing accuracy of machine learning algorithms*, European J. Appl. Math., (2020), https://doi.org/10.1017/S0956792520000224.

[45] A. Maurer, *The Rademacher complexity of linear transformation classes*, in Learning Theory, COLT 2006 (Pittsburgh, PA), Lecture Notes in Comput. Sci. 4005, Springer, Berlin, Heidelberg, 2006, pp. 65–78, https://doi.org/10.1007/11776420_8.

[46] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed., MIT Press, Cambridge, MA, 2018.

[47] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, *Exploring generalization in deep learning*, in Advances in Neural Information Processing Systems 30 (Long Beach, CA), Curran Associates, Red Hook, NY, 2017, pp. 5947–5956.

[48] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz, *Neural tangents: Fast and easy infinite neural networks in Python*, in 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, 2020, https://openreview.net/forum?id=SklD9yrFPS.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32 (Vancouver, Canada), Curran Associates, Red Hook, NY, 2019, pp. 8026–8037.

[50] B. Peherstorfer, K. Willcox, and M. Gunzburger, *Survey of multifidelity methods in uncertainty propagation, inference, and optimization*, SIAM Rev., 60 (2018), pp. 550–591, https://doi.org/10.1137/16M1082469.

[51] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis, *Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling*, Proc. R. Soc. A, 473 (2017), 20160751, https://doi.org/10.1098/rspa.2016.0751.

[52] P. Perdikaris, D. Venturi, J. O. Royset, and G. E. Karniadakis, *Multi-fidelity modelling via recursive co-kriging and Gaussian–Markov random fields*, Proc. R. Soc. A, 471 (2015), 20150018, https://doi.org/10.1098/rspa.2015.0018.

[53] N. Qian, *On the momentum term in gradient descent learning algorithms*, Neural Netw., 12 (1999), pp. 145–151, https://doi.org/10.1016/S0893-6080(98)00116-6.

[54] M. Raissi, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, J. Mach. Learn. Res., 19 (2018), pp. 932–955, https://dl.acm.org/doi/abs/10.5555/3291125.3291150.

[55] S. Reed, A. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. Freitas, *Parallel multiscale autoregressive density estimation*, in Proceedings of the 34th International Conference on Machine Learning (Sydney, Australia), PMLR, 2017, pp. 2912–2921.

[56] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, *ImageNet large scale visual*

recognition challenge, Int. J. Comput. Vis., 115 (2015), pp. 211–252, https://doi.org/10.1007/s11263-015-0816-y.

[57] V. Sindhwani, H. Q. Minh, and A. C. Lozano, *Scalable matrix-valued kernel learning for high-dimensional nonlinear multivariate regression and Granger causality*, in Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (Bellevue, WA), AUAI Press, 2013, pp. 586–595.

[58] M. Sun, X. Yan, and R. Sclabassi, *Solving partial differential equations in real-time using artificial neural network signal processing as an alternative to finite-element analysis*, in Proceedings of the International Conference on Neural Networks and Signal Processing (Nanjing, China), IEEE, Washington, DC, 2003, pp. 381–384, https://doi.org/10.1109/ICNNSP.2003.1279289.

[59] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, *Accelerating Eulerian fluid simulation with convolutional networks*, in Proceedings of the 34th International Conference on Machine Learning (Sydney, Australia), PMLR, 2017, pp. 3424–3433.

[60] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, *Generalizing from a few examples: A survey on few-shot learning*, ACM Comput. Surv., 53 (2020), 63, https://doi.org/10.1145/3386252.

[61] D. Weinshall, G. Cohen, and D. Amir, *Curriculum learning by transfer learning: Theory and experiments with deep networks*, in Proceedings of the 35th International Conference on Machine Learning (Stockholm, Sweden), PMLR, 2018, pp. 5238–5246.

[62] L. Zhang, J. Han, H. Wang, R. Car, and W. E, *Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics*, Phys. Rev. Lett., 120 (2018), 143001, https://doi.org/10.1103/PhysRevLett.120.143001.