META-PDE:
LEARNING TO SOLVE PDES QUICKLY WITHOUT A MESH

Tian Qin
School of Engineering and Applied Sciences
Harvard University
tqin@g.harvard.edu

Alex Beatson
Redesign Science
alex@redesignscience.com

Deniz Oktay
Department of Computer Science
Princeton University
doktay@princeton.edu

Nick McGreivy
Department of Astrophysical Sciences
Princeton University
mcgreivy@princeton.edu

Ryan P. Adams
Department of Computer Science
Princeton University
rpa@princeton.edu

ABSTRACT

Partial differential equations (PDEs) are often computationally challenging to solve, and in many settings many related PDEs must be be solved either at every timestep or for a variety of candidate boundary conditions, parameters, or geometric domains. We present a meta-learning based method which learns to rapidly solve problems from a distribution of related PDEs. We use meta-learning (MAML and LEAP) to identify initializations for a neural network representation of the PDE solution such that a residual of the PDE can be quickly minimized on a novel task. We apply our meta-solving approach to a nonlinear Poisson's equation, 1D Burgers' equation, and hyperelasticity equations with varying parameters, geometries, and boundary conditions. The resulting Meta-PDE method finds qualitatively accurate solutions to most problems within a few gradient steps; for the nonlinear Poisson and hyper-elasticity equation this results in an intermediate accuracy approximation up to an order of magnitude faster than a baseline finite element analysis (FEA) solver with equivalent accuracy. In comparison to other learned solvers and surrogate models, this meta-learning approach can be trained without supervision from expensive ground-truth data, does not require a mesh, and can even be used when the geometry and topology varies between tasks.

1 Introduction

Partial differential equations (PDEs) can be used to model many physical, biological, and mathematical systems. Such systems include those governing thermodynamics, continuum mechanics, and electromagnetism. Applications of PDEs outside science include modeling of traffic, populations, optimality of continuous control, and finance. Analytical solutions are rarely available for PDEs of practical importance; thus, computational methods must be used to find approximate solutions.

One of the most widely used approximation methods is finite element analysis (FEA). In FEA, the continuous problem is discretized and the solution is represented by a piecewise polynomial on a mesh. Solving PDEs with FEA can be computationally prohibitive, particularly when the problem geometry requires a fine mesh; the size of the system to be solved grows in proportion to the number of mesh cells. The main purpose of this paper is to use gradient-based meta-learning to accelerate solving PDEs with physics-informed neural networks (PINNs). This results in solvers that can achieve accurate solutions at reduced computational cost relative to FEA. Although these solvers have an initial training cost, they may provide computational savings in problems where a PDE must be solved repeatedly. Such problems could include parameter identification, design optimization, or in the solution of coupled time-dependent PDEs where, e.g., an elliptic equation is solved at each timestep.

PINNs use a neural network (NN) to represent the approximate solution of a PDE. The idea was popularized by Raissi et al. [24], and has been widely researched since. The key advantage of PINNs over traditional numerical solvers is that the PINN is able to provide a solution without the need to discretize the problem domain, i.e., the learned PDE solution is mesh-free. However, PINNs suffer from two major issues that limit their utility as forward solvers [28]. First, PINNs have not demonstrated the ability to solve all PDEs. In particular, PINNs tend to struggle to solve time-dependent PDEs whose solutions exhibit chaotic behavior or turbulent flow [27]. Second, vanilla PINNs tend to be dramatically slower than classic numerical methods. We attempt to mitigate this second issue by applying meta-learning to partially amortize the cost of optimization, thereby reducing the time required to find an accurate solution on a particular problem.

Forward-solving with PINNs requires optimization (i.e., learning); thus to accelerate forward solving we need to accelerate learning. Recent work in meta-learning has focused on how to construct learning algorithms that can adapt to a new task with as little additional training as possible. We focus on gradient-based meta learning algorithms such as MAML [9], REPTILE [17], and LEAP [10]. These algorithms view meta-learning as a bi-level optimization problem: the inner learning loop optimizes the model parameters for a given task, and the outer learning loop optimizes the inner loop's learning process across the tasks that the inner loop might encounter.

Main contribution: We introduce a framework which accelerates PDE solving by combining meta-learning and PINNs: Meta-PDE. PDE solving is accelerated by using gradient-based meta-learning techniques such as LEAP and MAML to train a PINN initialization which will converge quickly when optimized for a task drawn from a set of related tasks. The distribution of problems consists of different parameterizations of the PDE, such as different boundary conditions, initial conditions, the coefficients in the governing equation, or even the problem domain of the PDE. During deployment, the meta-learned model can be used to produce fast solutions to instances of PDEs in the distribution.

Our scheme has three important properties:

1. Training does not require supervised data provided by PDE solvers. This is in contrast to other learned PDE solvers and surrogate models, which typically train the solver to minimize the residual between the ground-truth solution and the predicted solution [3–5, 7, 13, 21, 29]. We instead minimize a residual of the governing equation (see section 2.1), eliminating the need for ground-truth data.

2. Meta-PDE is mesh-free and can be used on a broad class of boundary value problems, including problems with arbitrary geometries, and both time-dependent and time-independent PDEs.

3. Geometry, boundary conditions, and even the PDE are free to vary between tasks. Meta-PDE does not require a vector representation of the factor of variation between tasks which can be input to a neural network: instead, the user supplies an appropriate sampler for the domain and a loss function to measure the residual of the PDE solution. This differs from other learned PDE solvers and surrogate models, which are almost always trained for a single mesh or geometry and cannot be used when the geometry varies [11, 12, 14, 26].

Previous work has also explored meta-learning for PINNs. de Avila Belbute-Peres et al. [8] meta-trains a hyper-network that for each task can generate weights for a small neural network. The small neural network then becomes the approximate solution to the PDE. Psaros et al. [23] meta-learns a loss function which is used to optimize the NN; this is found to achieve performance benefits in comparison to both hand-crafted and online adaptive loss functions. Penwarden et al. [22] proposes a meta-learning approach which, like our Meta-PDE approach, learns an initialization of weights such that the NN can be optimized quickly. Penwarden et al. [22] compares MAML to other meta-learning approaches that initialize weights using a linear combination of basis functions. An important way in which these approaches are different from our MAML and LEAP-based approach is that the weight initialization depends on the task parameter that is varied. Penwarden et al. [22] finds that MAML achieves poor performance, only marginally better than random initialization; both our approach and the recently published Liu et al. [15] (which uses REPTILE instead of MAML) come to the opposite conclusion.

2 Meta-learning mesh-free PDE operators

We take our problems to be defined on the spatial domain $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$, and consider time-dependent PDEs

$$
\begin{aligned}
\frac{\partial}{\partial t} u(\boldsymbol{x}, t) + \mathcal{F}[u(\boldsymbol{x}, t)] &= 0 & \boldsymbol{x} \in \Omega, & \quad t \in [0, T], \\
\mathcal{G}(u)(\boldsymbol{x}, t) &= 0 & \boldsymbol{x} \in \partial\Omega, & \quad t \in [0, T], \\
u(\boldsymbol{x}, 0) &= \bar{u}_0(\boldsymbol{x}) & \boldsymbol{x} \in \Omega,
\end{aligned}
\tag{1}
$$

as well as time-independent PDEs which only have spatial dependence:

$$
\begin{aligned}
\mathcal{F}[u(\boldsymbol{x})] &= 0 & \boldsymbol{x} \in \Omega, \\
\mathcal{G}(u)(\boldsymbol{x}) &= 0 & \boldsymbol{x} \in \partial\Omega.
\end{aligned}
\tag{2}
$$

For time-dependent PDEs the time horizon is $[0, T]$. The function $u(\boldsymbol{x}, t)$ is the (unknown) solution to the PDE, while $\bar{u}_0(\boldsymbol{x})$ is the initial condition. In both cases $\mathcal{F}$ and $\mathcal{G}$ are governing equation and boundary operators that involve u and partial derivatives of u with respect to spatial coordinates $\boldsymbol{x}$.

2.1 Physics-Informed Neural Networks (PINN)

The goal of a PINN is to represent the approximate solution $u(\boldsymbol{x}, t)$ with a neural network $f_\theta(\boldsymbol{x}, t)$. Doing so requires learning $\theta \in \mathbb{R}^p$ such that f_θ approximates the solution to the PDE over the problem domain. Learning these parameters is an optimization problem, which requires defining a loss function whose minimum is the solution of the PDE. We choose a "physics-informed loss" which consists of an integral of the local residual of the differential equation over the problem domain as well as the boundary. Analytically, the residual should be integrated over the problem domain to compute the residual from satisfying the governing equation, and the initial condition and integrated over the boundary domain to compute the residual from satisfying the boundary conditions. For time-independent PDEs, this loss function is

$$
\mathcal{J}(u) = \int_\Omega ||\mathcal{F}(u)(\boldsymbol{x})||_2^2 \; dx + \int_{\partial\Omega} ||\mathcal{G}(u)(\boldsymbol{x})||_2^2 \; dx.
\tag{3}
$$

For time-dependent PDEs, this loss function is:

$$
\mathcal{J}(u) = \int_\Omega \left|\left| \frac{\partial}{\partial t} u(\boldsymbol{x}, t) + \mathcal{F}(u)(\boldsymbol{x}, t) \right|\right|_2^2 + ||u(\boldsymbol{x}, 0) - \bar{u}_0(\boldsymbol{x})||_2^2 \; dx + \int_{\partial\Omega} ||\mathcal{G}(u)(\boldsymbol{x}, t)||_2^2 \; dx.
\tag{4}
$$

During training, we randomly and uniformly sample collocation points from the PDE domain Ω and boundary $\partial\Omega$ and use these points $\mathcal{C} \in \Omega$ and $\partial\mathcal{C} \in \partial\Omega$ to form a Monte Carlo estimate of the true loss. For time-independent PDEs, this training loss is

$$
\mathcal{L}_{\text{PINN}}(f_\theta) = \frac{1}{|\mathcal{C}|} \sum_{\boldsymbol{x} \in \mathcal{C}} ||\mathcal{F}(f_\theta)(\boldsymbol{x})||_2^2 + \frac{1}{|\partial\mathcal{C}|} \sum_{\boldsymbol{x} \in \partial\mathcal{C}} ||\mathcal{G}(f_\theta)(\boldsymbol{x})||_2^2.
\tag{5}
$$

For time-dependent PDEs, the training loss is

$$
\begin{aligned}
\mathcal{L}_{\text{PINN}}(f_\theta) = &\frac{1}{|\mathcal{C}|} \sum_{\boldsymbol{x} \in \mathcal{C}} \left|\left| \frac{\partial}{\partial t} (f_\theta)(\boldsymbol{x}, t) + \mathcal{F}(f_\theta)(\boldsymbol{x}, t) \right|\right|_2^2 + \\
&\frac{1}{|\partial\mathcal{C}|} \sum_{\boldsymbol{x} \in \partial\mathcal{C}} ||\mathcal{G}(f_\theta)(\boldsymbol{x}, t)||_2^2 + \frac{1}{|\mathcal{C}|} \sum_{\boldsymbol{x} \in \mathcal{C}} ||f_\theta(\boldsymbol{x}, 0) - \bar{u}_0(\boldsymbol{x})||_2^2.
\end{aligned}
\tag{6}
$$

When the training converges, we expect f_θ to approximately satisfy the above equations, meaning that $\mathcal{L}_{\text{PINN}}(f_\theta)$ should be nearly zero.

2.2 Meta-PDE

Meta-PDE involves using gradient-based meta-learning to amortize the training time needed to fit f_θ on a problem drawn from a distribution of parameterized PDEs. We focus specifically on two meta-learning methods: LEAP [10] and MAML [9]. We describe LEAP-based Meta-PDE briefly here. MAML-based Meta-PDE is a straightforward extension and is described in Appendix A.

Most PDEs can be fully specified by their domain, boundaries, an operator representing governing equations, and an operator representing boundary conditions. When using Meta-PDE as a surrogate to compute an approximate solution to a given parametrization of the PDE (one task), the inputs to the Meta-PDE model imitate this general specification:

- A sampler $s(\Omega)$ which returns points in the domain Ω,
- A sampler $s(\partial\Omega)$ which returns points on the boundary $\partial\Omega$,
- An operator $\mathcal{F}$ representing governing equations,
- An operator $\mathcal{G}$ representing boundary conditions.

The operators $\mathcal{F}$ and $\mathcal{G}$ may be supplied directly and do not require a particular parametric form. The geometric dimension $\mathbb{R}^{d_\Omega}$ and solution dimension $\mathbb{R}^{d_u}$ must remain fixed across PDEs in the distribution, even though Ω is allowed to vary. The samplers and operators are sufficient to construct an estimator $\hat{\mathcal{L}}$ for the task loss $\mathcal{L}$ using Eqn.5 for time-independent problems and Eqn.6 time-dependent problems. Although $\hat{\mathcal{L}}(f)$ is unbiased as long as $s(\cdot)$ return points with uniform probability over their supports, note that unbiased estimation is not necessarily essential if we are interested in the case where $\mathcal{L}(f) = \hat{\mathcal{L}}(f) = 0$. Biased sampling will not change the minimizer of the energy estimator if we have a sufficiently expressive hypothesis class for f.

The LEAP-based Meta-PDE method learns the model initialization $\theta^0 \in \mathbb{R}^p$ for a neural network f_θ, which can then be trained to approximate the solution $u : \mathbb{R}^{d_\Omega} \to \mathbb{R}^{d_u}$ of an individual parametrization of the PDE. To learn θ^0, we start with a distribution of tasks, where each task represents a different parameterization of the PDE. Each task is specified by samplers and constraint operators for the boundary and loss. Then we draw a batch of n tasks with individual loss functions $\hat{\mathcal{L}}_i$, $i \in [n]$. The initialization for each inner task is θ^0, and is updated by the inner gradient update rule. During each inner gradient update, we update the meta-gradient per the LEAP algorithm. We unroll the inner learning loop K steps to find $f_{\theta_i^K}$: the approximate solution for each task i in the batch. After unrolling K update steps for n tasks, we update the learned model initialization θ^0 with the meta-gradient:

$$\theta^0 \leftarrow \theta^0 - \beta \nabla_{\theta^0} \sum_{i=1}^{n} \frac{1}{n} d(\theta^0; M_i), \tag{7}$$

where $d(\theta^0; M_i)$ is the distance of the gradient path for task i on its manifold M_i, as specified in Flennerhag et al. [10]. MAML involves a slightly different loss function and also learns step sizes for each parameter.

During deployment time, a "forward pass" computes an approximate solution for a given PDE parametrization with K steps of stochastic optimization. The K gradient steps minimize the training loss for the task $\mathcal{L}(f)$. If the model has been trained with LEAP-based Meta-PDE method, it will start from the meta-learned model initialization θ^0. If the model has been trained with MAML-based Meta-PDE method, it will start from the meta-learned model initialization θ^0 and the step size α will be also be specified for each parameter and each step:

$$\theta^k = \theta^{k-1} - \alpha \nabla_{\theta^{k-1}} \hat{\mathcal{L}}(f_{\theta^{k-1}}) \quad k = 1, \ldots, K . \tag{8}$$

In both cases, the Meta-PDE method returns the approximate solution f_{θ^K}, the neural network with the final set of parameters. One could further fine tune the model beyond K gradient steps to achieve higher solution accuracy at the cost of longer solving time.

3 Experiments

We evaluate the application of Meta-PDE to three example PDE problems: the nonlinear Poisson's equation, the 1D Burgers' equation, and the hyper-elasticity equation. We discuss the results of training on Burgers' equation in section 4, as well as in Appendix C. Meta-PDE methods are implemented in JAX [6]. In training, we sample points uniformly on the domain and the boundary. Our LEAP and MAML-based Meta-PDE models use a small NN with sinusoidal activation functions. The sinusoidal activation is initialized according to the scheme in Sitzmann et al. [25], although we replace $\omega_0 = 30.0$ in that paper with $\omega_0 = 3.0$ to avoid numerical issues when taking higher-order derivatives of a neural network's input-output function. Gradients in both inner and outer loop are clipped to have maximal norm 100.0. Additional NN hyperparameters are in Table 1 while training hyperparameters are in Table 2. We compare Meta-PDE's performance with a baseline FEniCS [2, 16] solver. We use the Mumps linear solver backend.

During deployment, the Meta-PDE solutions can be further improved by extending the number of "inner" training steps beyond what is used at training time. In our MAML-based Meta-PDE method for example, we train the meta-learned initialization using 5 inner-gradient steps. At deployment time, we can refine the Meta-PDE solution by using a greater number of inner-gradient steps. We compare the speed/accuracy trade-off achieved by varying the number of inner-gradient steps Meta-PDE takes during deployment with the speed/accuracy trade-off achieved by varying the resolution of the mesh used in FEA.

Table 1: Neural network hyperparameters for our Meta-PDE methods

PDE Problem	Meta-PDE Method	Hyperparameters					
		Num. of layers	Layer Size	Activation	Inner Steps	Inner LR	Outer LR
Nonlinear Poisson's		3				1.0×10^{-4}	1.0×10^{-5}
Burgers'	MAML	8	64	sin	5	1.0×10^{-4}	1.0×10^{-5}
Hyper-elasticity		5				1.0×10^{-5}	5.0×10^{-6}
Nonlinear Poisson's		5	64		60	2.5×10^{-5}	5.0×10^{-5}
Burgers'	LEAP	10	128	sin	80	1.0×10^{-6}	5.0×10^{-5}
Hyper-elasticity		10	128		20	5.0×10^{-6}	5.0×10^{-6}

Table 2: Training hyperparameters for our meta-PDE methods and training time on one NVIDIA T4 GPU

PDE Problem	Meta-PDE Method	Hyperparameters					
		Batch Size	Sampled Points	Iterations	Training Time	Inner Optimizer	Outer Optimizer
Nonlinear Poisson's			2048	120,000	4 hrs		
Burgers'	MAML	8	1024	60,000	11 hrs	SGD	Adam
Hyper-elasticity			1024	180,000	21 hrs		
Nonlinear Poisson's			4096	55,000	5 hrs		
Burgers'	LEAP	8	2048	7,000	7 hrs	Adam	Adam
Hyper-elasticity			1024	140,000	8 hrs		

3.1 Nonlinear Poisson's Equation

Poisson's equation is one of the most ubiquitous equations in physics. For example, the linear Poisson equation calculates the electrostatic or gravitational field caused by electric charges or mass particles. Solving systems of coupled time-dependent PDEs often requires the solution of a Poisson equation at each timestep [18]. Since the linear Poisson's equation can be solved analytically, we demonstrate Meta-PDE on a nonlinear Poisson problem with varying source terms, boundary conditions, and geometric domain. This nonlinear Poisson's equation takes the form

$$\nabla \cdot \left[(1 + 0.1u^2)\nabla u(\boldsymbol{x})\right] = f(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega$$
$$u(\boldsymbol{x}) = b(\boldsymbol{x}) \qquad \boldsymbol{x} \in \partial\Omega$$

where $u \in \mathbb{R}^1$ and $\Omega \subset \mathbb{R}^2$. Using our notation from the previous section, this is equivalent to constraining the solution in the domain with an operator $\mathcal{F}(u) = ((1 + 0.1u^2)\nabla u) - f$, and constraining the solution on the boundary with an operator $\mathcal{G}(u) = u - b$.

The domain Ω is a disc-like shape centered at the origin, defined in polar coordinates with varying radius about the origin $r(\theta) = r_0[1 + c_1 \cos(4\theta) + c_2 \cos(8\theta)]$, where the varying parameters are $c_1, c_2 \sim \mathcal{U}(-0.2, 0.2)$. The source term f is a sum of radial basis functions $f(\boldsymbol{x}) = \sum_{i=1}^3 \beta_i \exp \|\boldsymbol{x} - \mu_i\|_2^2$, where $\beta_i \in \mathbb{R}^1$ and $\mu_i \in \mathbb{R}^2$ are both drawn
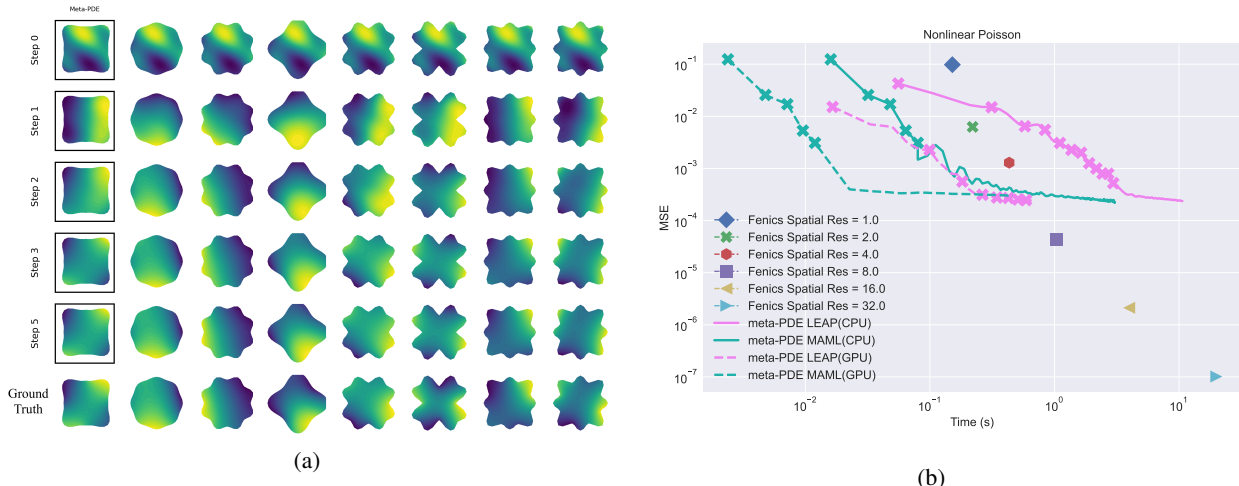
Figure 1: (a) Solutions to nonlinear Poisson's equations with varying domains, boundary conditions, and source terms. First Row: Solution represented by Meta-PDE initial NN parameters. Second Row Onwards: Solution after each gradient step in the Meta-PDE inner loop. Bottom: Ground truth FE solution. (b) Speed/Accuracy trade-off for Meta-PDE and FEA. The x-axis is time to solve and y-axis is accuracy, as measured by MSE. For Meta-PDE, we vary the number of training steps after deployment. For FE, we vary the mesh resolutions. Overall, Meta-PDE yields comparable speed/accuracy performance on CPU at intermediate accuracy but better performance on GPU. Meta-PDE reaches an accuracy ceiling at an MSE between 10^{-3} and 10^{-4}, while the FEA solution may be refined to higher accuracy by increasing the resolution of the mesh.

from standard normal distributions. The boundary condition b is a periodic function, defined in polar coordinates as $b(x) = b_0 + b_1 \cos(\theta) + b_2 \sin(\theta) + b_3 \cos(\theta) + b_4 \sin(\theta)$, where the parameters $b_{0:4} \sim \mathcal{U}(-1, 1)$.

Figure 1a shows the ground truth (baseline) solution for eight PDE problems used in the validation set. The same figure also shows the MAML meta-learned initialization, which can quickly adapt to each PDE problems in five gradient steps.

Figure 1b shows the mean squared error (MSE) of each solution and solving time required for the desired accuracy. For the Fenics solution, we vary the mesh resolution (Spatial Res in Figure 1b). For the Meta-PDE solution, we increase the number of training steps, starting from the meta-learning initialization. For MAML-based Meta-PDE method, we also start from the meta-learned step size. The highest-resolution FEA method was taken as ground truth and was used to compute MSE. The MSE and solving times were evaluated using 8 held-out problems from the same distribution, and the 8 held-out problems were not used during training. Mean-squared errors are computed between the value of a given approximate solution and the value of the ground truth (highest resolution finite element solution) at 1024 randomly sampled points within the domain. The held-out set configuration remains the same for the other two experiments below.

Meta-PDE learns to efficiently output moderately accurate solutions. When run on the same CPU (3.6 GHz Intel Xeon Platinum 8000 series) it is about $1-2\times$ faster than a finite element method with similar accuracy. Unlike finite element models, Meta-PDE can be easily accelerated by a GPU, and on GPU we see close to $50\times$ speed up in deployment over similar accuracy CPU-based finite element models.

3.2 Hyper-Elasticity Equation

Hyper-elastic materials undergo large shape deformation when force is applied and the stress-strain relation for those materials are highly nonlinear. Rubber is a common example of hyper-elastic materials. the Hyper-elasticity equation models the deformation of those rubber-like materials under different external forces. In particular, we model a homogeneous and isotropic hyper-elastic material under deformation when compressed uniaxially. We assume no additional body or traction force applied to the structure. The goal is to model the final deformation displacement u, which maps the material position change from the initial reference position $\mathbf{X}$ to its current deformed location x.

There are two different approaches to encode the loss function for the hyper-elasticity equation. First, one could directly minimize the residual term in the original strong form, the same approach as we have done for the nonlinear Poisson's equation and for the Burgers' equation. For example, Abueidda et al. [1] has used the first approach to encode the loss
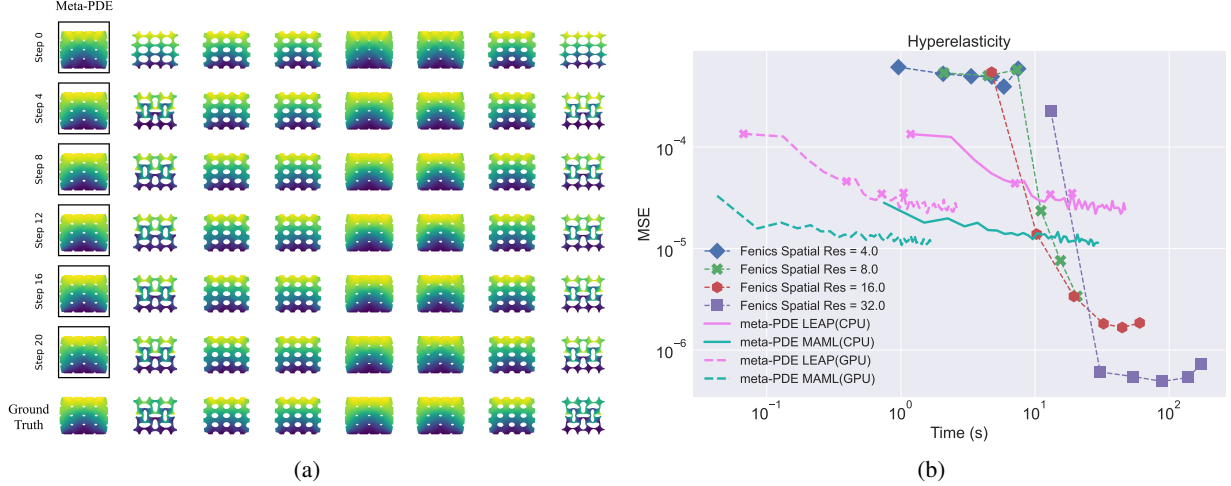
Figure 2: (a) Solutions to hyper-elasticity equations with varying domains. First Row: Solution represented by Meta-PDE initial network parameters. Second Row Onwards: Solution after each gradient step in the Meta-PDE inner loop. Bottom: Ground truth FE solution. (b) Speed/Accuracy trade-off for Meta-PDE and FEA. The x-axis is time to solve and y-axis is accuracy, as measured by MSE. For Meta-PDE, we vary the number of training steps after deployment. For FE, we vary the mesh and boundary resolution. Meta-PDE yields a better speed/accuracy trade-off at intermediate accuracy but is not able to efficiently reach very high accuracy.

function for PINN. Alternatively, one can minimize the Helmholtz free energy of the system and find the corresponding minimizer u. We use the second approach to solve the hyper-elasticity equation. See Appendix B for details of the PDE formulation and loss function derivation.

We consider the deformation of a two-dimensional porous hyper-elastic material under compression. Their material properties could be very different from their solid counterparts. Because of these interesting differences, the hyper-elastic behavior of porous structures is an active field of research in material science [19, 20]. Following the problem setup in [19], we use the following parametrized equations to model the shape of the pores:

$$x_1 = r(\theta) \cos \theta, \ x_2 = r(\theta) \sin \theta$$
$$r(\theta) = r_0 \left[1 + c_1 \cos(4\theta) + c_2 \cos(8\theta) \right]$$
$$r_0 = \frac{L_0 \sqrt{2\phi_0}}{\sqrt{\pi \left(2 + x_1^2 + x_2^2 \right)}}$$

ϕ_0 is the initial porosity of the structure, and it is sampled uniformly: $\phi_0 \sim \mathcal{U}(0.0, 0.75)$ in our experiment setup. The parameter pair (c_1, c_2) determines the shape of the pore, and we fixed them to $(0.0, 0.0)$ so that we only work with the circular porous shape. L_0 is the initial center-to-center distance between neighboring pores. We also fixed the distance L_0 so we work with fixed number of pores on the given material size. With the pore shape and the distance between pore centers L_0 fixed, the size of the pore determines the porosity of the structure. The porosity of the structure affects the macroscopic deformation behavior of the structure. Figure 2a shows the ground truth (baseline) solution for eight PDE problems used in the validation set. The same figure also shows the LEAP meta-learned initialization, which can quickly adapt to each PDE problems in 20 gradient steps.

Figure 2b shows the mean squared error (MSE) of each solution and solving time required for the desired accuracy. Meta-PDE learns to output accurate solutions, and when run on the same CPU is about $5 - 10\times$ faster than a finite element method with similar accuracy. Running Meta-PDE on a GPU gives close to $100\times$ speed up in deployment over a similar accuracy finite element model run on CPU.
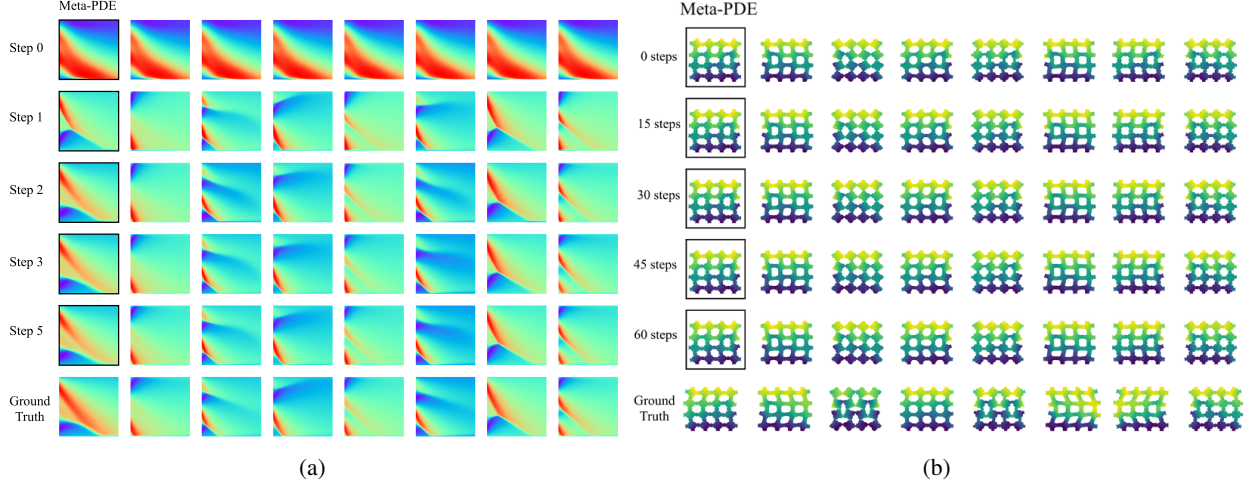
7

Figure 3: Problems instances where Meta-PDE has difficulty. (a) Solutions to Burgers's Equations with varying initial conditions, and boundary conditions. First Row: Solution represented by Meta-PDE initial NN parameters. Second Row Onwards: Solution after each gradient step in the Meta-PDE inner loop. Bottom: Ground truth FEA solution. Although the Meta-PDE method achieves qualitatively accurate results after a few gradient steps, and is faster than a naive application of FEA, the meta-solver is still slower than a well-chosen baseline for Burgers' equation. (b) Solutions to hyper-elasticity equations with varying porous shape. Top Row: solution represented by Meta-PDE initial neural network parameters. Second row onwards: solution after each gradient step in the Meta-PDE inner loop. Bottom: Ground truth FEA solution. Looking at the bottom two rows, we can see that on many problem instances Meta-PDE fails to find an accurate solution.

4 Discussion

Meta-PDE Methods Comparison MAML-based Meta-PDE outperforms LEAP-based Meta-PDE during deployment time in accuracy for a given runtime, while requiring less hyperparameter tuning during training. We believe that this superior performance is due to MAML having a meta-learned per-state per-parameter step size. The advantage of LEAP-based Meta-PDE lies in the meta-training process: LEAP is faster to train and uses less memory than MAML, which required checkpointing for some PDEs.

Task Domain Generalizability In our study, we restrict each meta-learner to one type of PDE, and for each type of PDE we define the distribution of related tasks via different parameterizations of the same PDE type. As we increase the volume of this distribution, either by increasing the range of parameters or by increasing the number of parameters, the meta-learning tasks becomes harder. As a result, we need to use a larger network architecture, increase the number of inner training steps, and increase the meta-learner's training time to allow it to converge. The quality of the final model also deteriorates. To illustrate this, we looked at a how different pore shapes affect the macroscopic behavior of the material, which is the hyper-elasticity problem studied by Overvelde and Bertoldi [19]. Following the set up in Section 3.2, we fix the initial porosity of the structure $\phi_0 = 0.5$. We now vary the pore shape by sampling c_1 and c_2 in the parameter pair (c_1, c_2) from a uniform distribution: $c_{1,2} \sim \mathcal{U}(-0.4, 0.4)$. The initial center-to-center distance between neighboring pore L_0 is fixed like before. In this setup, the shape of the pore determines the macroscopic deformation behavior of the structure. As we enlarge the range of possible shapes by increasing the range that we draw possible (c_1, c_2) value from, the accuracy of the solutions produced by Meta-PDE starts to degrade. Figure 3b shows that for many pore shapes, Meta-PDE finds an incorrect solution.

Easy-to-solve PDEs and Hard-to-solve PINNs In addition to our experiments in section 3 we used Meta-PDE to train a meta-solver for the 1D Burgers' equation. Our distribution of tasks consists of varying initial and boundary conditions. The trained meta-solver was able to consistently achieve accurate results with 5 or fewer gradient steps (see fig. 3a). Although our meta-solver was successful in this regard, and was more efficient than a baseline FEA method written in FEniCS, it was not able to compete with an efficient finite volume method with Godunov flux written in JAX. This suggests that a naive application of Meta-PDE is not well suited for cases where nonlinear FEA under-performs compared to another simple baseline. Appendix C and tables 1 and 2 contain a full explanation of the distribution of problems and training hyperparameters for the Burgers' equation experiment.

We also tried applying Meta-PDE to the Navier-Stokes equations and the 2D Burger's equation. Here, Meta-PDE could not find reasonable solutions for a non-trivial distribution of tasks. PINNs generally find these time-dependent PDEs hard to solve, so it is not surprising that our meta-learning approach failed to learn how to solve them. However, recent work [28] has found that by modifying the loss function to better respect the principle of causality in time-dependent PDEs, hard-to-solve PDEs such as the Navier-Stokes equations can be successfully solved with PINNs. We are interested in seeing whether incorporating this modified loss function into our meta-learning approach could help us meta-learn solvers for these PDEs.

5 Conclusion

Meta-PDE uses meta-learning to amortize PDE solving by accelerating optimization of a PINN representation of the solution. Unlike other fast surrogate models (but like PINNs) our method is mesh-free and data-free, a desirable property when geometry is complex and/or varying across problems within the class we wish to amortize. Unlike PINNs, which are generally too slow to be competitive even with FEA, our method improves on the Pareto frontier of computational cost vs accuracy over FEA. We apply our method to amortize solving of PDEs with varying and complex geometries and terms: non-linear Poisson's equations, hyper-elasticity equations and a 1D Burgers equation. After meta-training, our method both (a) achieves qualitatively correct results for most problems in the distribution and (b) achieves these results after only a few gradient steps, resulting in a solver that is between 1 and 10 times faster than our FEniCS baseline.

This method has some caveats. First, our meta-solvers take a long time to train—several hours on a GPU. Second, our meta-solvers do not have the convergence guarantees that come with methods such as FEA. Third, meta-PDE appears to be better suited for time-independent (i.e., elliptic) PDEs, rather than time-dependent (i.e., hyperbolic) PDEs where information travels along characteristics. For example, we apply Meta-PDE to the 1D Burger's equation, and although we achieve qualitatively accurate results in a few gradient steps, our meta-solver is slower than a strong JAX baseline using the finite volume method with Godunov flux. Finally, there is a vast world of difficult-to-solve PDEs which require specially tailored computational tricks to solve (whether with PINNs or FEA) due to structure in the governing equations, and in this paper we consider only some relatively simple example PDEs. Despite these caveats, we believe that meta-PDE provides a generic and compelling approach to accelerated solving of PDEs with challenging domain geometries without ground-truth data or mesh.

6 Acknowledgements

References

[1] Diab W Abueidda, Qiyue Lu, and Seid Koric. Meshless physics-informed deep learning method for three-dimensional solid mechanics. *International Journal for Numerical Methods in Engineering*, 122(23):7182–7201, 2021.

[2] Martin S. Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E. Rognes, and Garth N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015. doi: 10.11588/ans.2015.100.20553.

[3] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. doi: 10.1073/pnas.1814058116. URL https://www.pnas.org/doi/abs/10.1073/pnas.1814058116.

[4] Alex Beatson, Jordan Ash, Geoffrey Roeder, Tianju Xue, and Ryan P Adams. Learning composable energy surrogates for pde order reduction. *Advances in Neural Information Processing Systems*, 33, 2020.

[5] Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. *Journal of Computational Physics*, 398:108910, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.108910. URL https://www.sciencedirect.com/science/article/pii/S0021999119306151.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[7] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2022. URL https://arxiv.org/abs/2202.03376.

[8] Filipe de Avila Belbute-Peres, Yi-fan Chen, and Fei Sha. HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks. In *The Symbiosis of Deep Learning and Differential Equations*, 2021.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[10] Sebastian Flennerhag, Pablo G Moreno, Neil D Lawrence, and Andreas Damianou. Transferring knowledge across learning processes. *arXiv preprint arXiv:1812.01054*, 2018.

[11] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid PDE solvers. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2415–2423. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/greenfeld19a.html.

[12] Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees, 2019. URL https://arxiv.org/abs/1906.01200.

[13] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118 (21):e2101784118, 2021. doi: 10.1073/pnas.2101784118. URL https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

[14] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[15] Xu Liu, Xiaoya Zhang, Wei Peng, Weien Zhou, and Wen Yao. A novel meta-learning initialization method for physics-informed neural networks. *Neural Computing and Applications*, pages 1–24, 2022.

[16] Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012. doi: 10.1007/978-3-642-23099-8.

[17] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[18] Steven A Orszag. Numerical methods for the simulation of turbulence. *The Physics of Fluids*, 12(12):II–250, 1969.

[19] Johannes TB Overvelde and Katia Bertoldi. Relating pore shape to the non-linear response of periodic elastomeric structures. *Journal of the Mechanics and Physics of Solids*, 64:351–366, 2014.

[20] Johannes Tesse Bastiaan Overvelde, Sicong Shan, and Katia Bertoldi. Compaction through buckling in 2d periodic, soft and porous structures: effect of pore shape. *Advanced Materials*, 24(17):2337–2342, 2012.

[21] Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations, 2020. URL https://arxiv.org/abs/2010.00072.

[22] Michael Penwarden, Shandian Zhe, Akil Narayan, and Robert M Kirby. Physics-informed neural networks (PINNs) for parameterized PDEs: A metalearning approach. *arXiv preprint arXiv:2110.13361*, 2021.

[23] Apostolos F Psaros, Kenji Kawaguchi, and George Em Karniadakis. Meta-learning PINN loss functions. *Journal of Computational Physics*, 458:111121, 2022. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2022.111121. URL https://www.sciencedirect.com/science/article/pii/S0021999122001838.

[24] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[25] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[26] Kimberly Stachenfeld, Drummond B. Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation, 2021. URL https://arxiv.org/abs/2112.15275.

[27] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022. URL https://arxiv.org/abs/2203.07404.

[28] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022. URL `https://arxiv.org/abs/2203.07404`.

[29] Jiawei Zhuang, Dmitrii Kochkov, Yohai Bar-Sinai, Michael P. Brenner, and Stephan Hoyer. Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. *Phys. Rev. Fluids*, 6:064605, Jun 2021. doi: 10.1103/PhysRevFluids.6.064605. URL `https://link.aps.org/doi/10.1103/PhysRevFluids.6.064605`.

A MAML Based Meta-PDE

The MAML-based Meta-PDE method learns the model initialization θ^0 for a neural network f_θ and also learns the learning rate α for each parameter at each inner step. To learn θ^0 we start with a distribution of tasks, where each task represents a different parameterization of the PDE. Then we draw a batch of n tasks with individual loss functions $L_i, i \in [n]$. The initialization for each inner task is θ^0. The gradient update rule for the inner task is simply SGD:

$$\theta_i^j = \theta_i^{j-1} - \alpha \nabla_\theta \hat{\mathcal{L}}_i(f_\theta) \quad j \in \{1, 2, 3, ..., K\}$$

We unroll the inner learning loop K steps to find $f_{\theta_i^K}$: the approximate solution for each task i in the batch. The meta-loss is the average loss for those n tasks:

$$\mathcal{L}_{\text{MAML}} = \frac{1}{n} \sum_{i=1}^{n} \hat{\mathcal{L}}_i(f_{\theta_i^K})$$

We perform backpropagation through the inner loop to find the gradients w.r.t meta-initialization θ^0 and use the gradients to update θ^0 in the outer loop training:

$$\theta^0 \leftarrow \theta^0 - \beta \frac{1}{n} \nabla_{\theta^0} \sum_{i=1}^{n} \hat{\mathcal{L}}_i(f_{\theta_i^K}) \tag{9}$$

We also perform backpropagation through the inner loop to find the gradients w.r.t. the per-step, per-parameter step size α and use the gradients to update the α in the outer loop training:

$$\alpha \leftarrow \alpha - \beta \frac{1}{n} \nabla_\alpha \sum_{i=1}^{n} \hat{\mathcal{L}}_i(f_{\theta_i^K}) \tag{10}$$

Eqn. 7 defines the meta-gradient for the LEAP-based Meta-PDE method. Eqn. 9, 10 defines the meta-gradient for the MAML-based Met-PDE method.

B Hyper-Elasticity Equation Details

The material's initial reference position is $\mathbf{X}$ and its current deformed location is $\boldsymbol{x}$. The unknown function u maps the material's position change from the initial reference position $\mathbf{X}$ to its current deformed location $\boldsymbol{x}$:

$$u = \boldsymbol{x} - \mathbf{X}$$

The deformation gradient F is defined as

$$F \equiv \frac{\partial \boldsymbol{x}}{\partial \mathbf{X}} = \frac{\partial}{\partial \mathbf{X}} (\mathbf{X} + u) \equiv \frac{\partial \mathbf{X}}{\partial \mathbf{X}} + \frac{\partial u}{\partial \mathbf{X}} = \mathbf{I} + \frac{\partial u}{\partial \mathbf{X}}$$

The constitutive law in continuum mechanics relates Piola-Kirchhoff stress P with deformation gradient F using the following relations:

$$P = \frac{\partial \psi}{\partial F},$$

where ψ is the Helmholtz free energy. For a Neo-Hookean hyperelastic material in 2-D, the energy is given by:

$$\psi = \frac{1}{2} \lambda \left(\log(J)\right)^2 - \mu \log(J) + \frac{\mu}{2} (\mathbf{I}_c - 2).$$

There are two invariants in the above equation. The first is $\mathbf{I}_c \equiv \text{tr}(C)$ and C is the right Cauchy-Green tensor, defined as $C = F^T F$. The second invariant is $J \equiv \det(F)$. Substitute the two invariant into the above equation, the Piola-Kirchohoff stress P becomes:

$$P = \frac{\partial \psi(F)}{\partial F} = \mu F \left(\lambda \ln(J) - \mu\right) F^{-T}$$

In the absence of body and traction forces, the Hyperelasticity equations can be written as

$$\begin{aligned} \nabla_{\mathbf{X}} \cdot P &= 0 & \mathbf{X} \in \Omega \\ \hat{u} &= g(u) & \mathbf{X} \in \Gamma_u \\ P \cdot N &= T & \mathbf{X} \in \Gamma_T. \end{aligned}$$
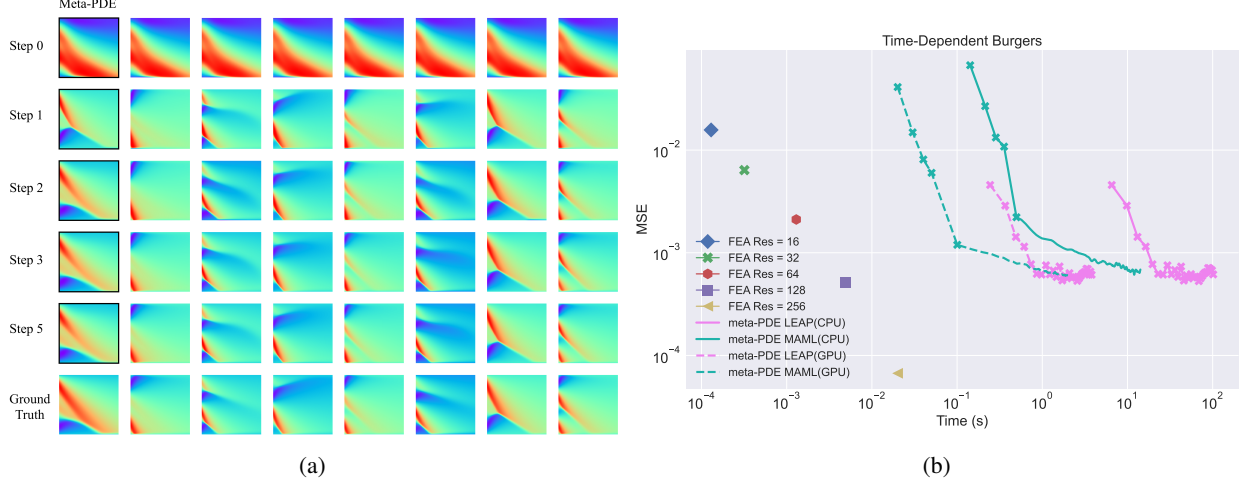
Figure 4: (a) Solutions to Burgers's Equations with varying initial conditions and boundary conditions. First Row: Solution represented by Meta-PDE initial NN parameters. Second Row Onwards: Solution after each gradient step in the Meta-PDE inner loop. Bottom: Ground truth FEA solution (b) Speed/Accuracy trade-off for Meta-PDE and FEA. The x-axis is time to solve and y-axis is accuracy, as measured by MSE. For Meta-PDE, we vary the number of training steps after deployment. For FE, we vary the mesh resolution and number of timesteps.

N is the normal vector relative to $\mathbf{X}$. Because the traction force is absent, we set $T = 0$ on boundary Γ_T. Dirichlet boundary conditions are imposed on boudnary Γ_u.

The solution u to the Hyperelasticity equations also acts as the minimizer of the Helmholtz free energy Π of the entire system:

$$u = \text{argmin}_u \, \Pi(u) = \text{argmin}_u \left[\int_\Omega \psi \, d\boldsymbol{x} \right]$$

During training, we randomly and uniformly sample collocation points from the PDE domain Ω and Dirichlet boundary Γ_u and use these points $\mathcal{C} \in \Omega$ and $\partial\mathcal{C} \in \Gamma_u$ to form a Monte Carlo estimate of the true loss. For the Hyperelasticity equations, the training loss is

$$\mathcal{L}_{\text{PINN}}(f_\theta) = \frac{1}{|\mathcal{C}|} \sum_{\boldsymbol{x} \in \mathcal{C}} \psi(f_\theta) + \frac{1}{|\partial\mathcal{C}|} \sum_{\boldsymbol{x} \in \partial\mathcal{C}} ||\hat{u} - g(f_\theta)||_2^2 \, .$$

C Burger's Equation

Burger's equation is a time-dependent PDE that models a system consisting of a moving viscous fluid. The 1D version of the equation models the fluid flow through an ideal thin pipe. The strong form of Burger's equation is given by

$$\begin{aligned}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} &= 0, & x \in \Omega, & \quad t \in [0, T] \\
u(x, 0) &= u_0(x), & x \in \Omega \\
u(x, t) &= \bar{u}, & x \in \partial\Omega, & \quad t \in (0, T] \, .
\end{aligned}$$

The unknown $u(x, t)$ is the speed of the fluid. If the viscosity ν is low, the fluid develops a shock wave. Following the derivation in Eqn. 6, the equation is constrained in the domain with the operator $\mathcal{F} = u \cdot \partial u/\partial x - \nu \cdot \partial^2 u/\partial x^2$, and constrained on the domain boundary with the operator $\mathcal{G} = u(x, 0) - u_0(x)$.

We look at the wave formation inside an ideal thin pipe with unit length: $x \in [0, 1]$. We also constrain the time horizon $t \in [0, 1]$. The initial condition is defined by three sinusoidal functions $u(x) = \sin(\pi x) + \theta_1 \sin(2\pi x) + \theta_2 \sin(4\pi x)$. The varying parameters are $\theta_1, \theta_2 \sim \mathcal{U}(-2.0, 2.0)$. Dirichlet boundary conditions are imposed on both the left boundary $x = 0$ and the right boundary $x = 1$, and are both set to $\bar{u} = 0$. The viscosity ν is set to 0.01.

For the ground truth comparison, we first used a baseline FEniCS solver with implicit Euler for the time integration. The Meta-PDE method outperformed the Fenics baseline by $10 - 20\times$ in speed when executed on the same CPU. However,

13

we found that the FEniCS baseline solver was significantly slower than a finite volume method with Godunov Flux and explicit RK3 timestepping written in JIT-compiled JAX. Figure 4a shows the ground truth (baseline) solution for the eight PDE problems used in the validation set. The same figure also shows the MAML meta-learned initialization, which can quickly adapt to each PDE problems in 5 gradient steps. Figure 4b shows the speed and accuracy for the eight PDE problems solved by the fast finite volume method. Resolution indicates the mesh resolution for the spatial domin and the temporal resolution is fixed to be 10 times the corresponding mesh resolution. We see that Meta-PDE learns to output accurate solutions, but when compared with the fast finite volume method, the Meta-PDE method is slower than the finite element method with similar accuracy.