Principled Acceleration of Iterative Numerical Methods Using Machine Learning

Sohei Arisaka [1 2] **Qianxiao Li** [1]

Abstract

Iterative methods are ubiquitous in large-scale scientific computing applications, and a number of approaches based on meta-learning have been recently proposed to accelerate them. However, a systematic study of these approaches and how they differ from meta-learning is lacking. In this paper, we propose a framework to analyze such learning-based acceleration approaches, where one can immediately identify a departure from classical meta-learning. We theoretically show that this departure may lead to arbitrary deterioration of model performance, and at the same time, we identify a methodology to ameliorate it by modifying the loss objective, leading to a novel training method for learning-based acceleration of iterative algorithms. We demonstrate the significant advantage and versatility of the proposed approach through various numerical applications.

1. Introduction

It is common and important in science and engineering to solve similar computational problems repeatedly. For example, in an actual project of structural engineering, the structure design is considered iteratively to satisfy various constraints, such as safety, cost, and building codes. This iterative design process often involves a large number of structural simulations (Gallet et al., 2022). Another example is systems biology, where an important but challenging problem is to estimate parameters of mathematical models for biological systems from observation data, and solving this inverse problem often requires many numerical simulations (Moles et al., 2003; Chou & Voit, 2009). In these situations, we can utilize the data of the previously solved problems to solve the next similar but unseen problems more efficiently, and machine learning is a natural and effective approach for this.

[1]Department of Mathematics, National University of Singapore [2]Kajima Corporation, Japan. Correspondence to: Sohei Arisaka <sohei.arisaka@u.nus.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

Thus, in recent years, many learning-based methods have been proposed for repeated solutions of computational problems. These range from the direct prediction of solutions as a supervised learning problem (Guo et al., 2016; Tang et al., 2017; Shan et al., 2020; Özbay et al., 2021; Cheng et al., 2021; Pfaff et al., 2021; Li et al., 2021) to tightly coupling machine learning and traditional scientific computing to take advantage of both (Ajuria Illarramendi et al., 2020; Um et al., 2020; Huang et al., 2020; Vaupel et al., 2020; Luna et al., 2021; Nikolopoulos et al., 2022). This paper focuses on the acceleration of iterative algorithms by (meta-)learning (Feliu-Fabà et al., 2020; Venkataraman & Amos, 2021; Liu et al., 2022; Huang et al., 2022; Guo et al., 2022; Chen et al., 2022; Psaros et al., 2022), which belongs to the second class. For instance, (Chen et al., 2022) uses meta-learning to generate smoothers of the Multi-grid Network for parametrized partial differential equations (PDEs), and (Guo et al., 2022) proposes a meta-learning approach to learn effective solvers based on the Runge-Kutta method for ordinary differential equations. In (Liu et al., 2022; Huang et al., 2022; Psaros et al., 2022), meta-learning is used to accelerate the training of physics-informed neural networks for solving PDEs.

However, to date, there lacks a systematic scrutiny of the overall approach. For example, does directly translating a meta-learning algorithm, such as gradient-based meta-learning (Hospedales et al., 2021), necessarily lead to acceleration in iterative methods? In this work, we perform a systematic framework to study this problem, where we find that there is indeed a mismatch between currently proposed training methods and desirable outcomes for scientific computing. Using numerical examples and theoretical analysis, we show that minimizing the solution error does not necessarily speed up the computation. Based on our analysis, we propose a novel and principled training approach for learning-based acceleration of iterative methods along with a practical loss function that enables gradient-based learning algorithms.

Our main contributions can be summarized as follows:

1. In Section 2, we propose a general framework, called gradient-based meta-solving, to analyze and develop learning-based numerical methods. Using the framework, we reveal the mismatch between the training

and testing of learning-based iterative methods in the literature. We show numerically and theoretically that the mismatch actually causes a problem.

2. In Section 3, we propose a novel training approach to directly minimize the number of iterations along with a differentiable loss function for this. Furthermore, we theoretically show that our approach can perform arbitrarily better than the current one and numerically confirm the advantage.
3. In Section 4, we demonstrate the significant performance improvement and versatility of the proposed method through numerical examples, including nonlinear differential equations and nonstationary iterative methods.

2. Problem Formulation and Analysis of Current Approaches

Iterative solvers are powerful tools to solve computational problems. For example, the Jacobi method and SOR (Successive Over Relaxation) method are used to solve PDEs (Saad, 2003). In iterative methods, an iterative function is iteratively applied to the current approximate solution to update it closer to the true solution until it reaches a criterion, such as a certain number of iterations or error tolerance. These solvers have parameters, such as initial guesses and relaxation factors, and solver performance is highly affected by the parameters. However, the appropriate parameters depend on problems and solver configurations, and in practice, it is difficult to know them before solving problems.

In order to overcome this difficulty, many learning-based iterative solvers have been proposed in the literature (Hsieh et al., 2018; Um et al., 2020; Stanziola et al., 2021; Chen et al., 2022; Kaneda et al., 2022; Azulay & Treister, 2022; Nikolopoulos et al., 2022). However, there lacks a unified perspective to organize and understand them. Thus, in Section 2.1, we first introduce a general and systematic framework for analyzing and developing learning-based numerical methods. Using this framework, we identify a problem in the current approach and study how it degrades performance in Section 2.2.

2.1. General Formulation of Meta-solving

Let us first introduce a general framework, called meta-solving, to analyze learning-based numerical methods in a unified way. We fix the required notations. A task τ is any computational problem of interest. Meta-solving considers the solution of not one but a distribution of tasks, so we consider a task space $(\mathcal{T}, P)$ as a probability space that consists of a set of tasks $\mathcal{T}$ and a task distribution P. A loss function $\mathcal{L} : \mathcal{T} \times \mathcal{U} \to \mathbb{R} \cup \{\infty\}$ is a function to measure how well the task τ is solved, where $\mathcal{U}$ is the solution space in which we find a solution. To solve a task τ means to find an approximate solution $\hat{u} \in \mathcal{U}$ which minimizes $\mathcal{L}(\tau, \hat{u})$. A solver Φ is a function from $\mathcal{T} \times \Theta$ to $\mathcal{U}$. $\theta \in \Theta$ is the parameter of Φ, and Θ is its parameter space. Here, θ may or may not be trainable, depending on the problem. Then, solving a task $\tau \in \mathcal{T}$ by a solver Φ with a parameter θ is denoted by $\Phi(\tau; \theta) = \hat{u}$. A meta-solver Ψ is a function from $\mathcal{T} \times \Omega$ to Θ, where $\omega \in \Omega$ is a parameter of Ψ and Ω is its parameter space. A meta-solver Ψ parametrized by $\omega \in \Omega$ is expected to generate an appropriate parameter $\theta_\tau \in \Theta$ for solving a task $\tau \in \mathcal{T}$ with a solver Φ, which is denoted by $\Psi(\tau; \omega) = \theta_\tau$. When Φ and Ψ are clear from the context, we write $\Phi(\tau; \Psi(\tau; \omega))$ as $\hat{u}(\omega)$ and $\mathcal{L}(\tau, \Phi(\tau; \Psi(\tau; \omega)))$ as $\mathcal{L}(\tau; \omega)$ for simplicity. Then, by using the notations above, our meta-solving problem is defined as follows:

Definition 2.1 (Meta-solving problem)**.** For a given task space $(\mathcal{T}, P)$, loss function $\mathcal{L}$, solver Φ, and meta-solver Ψ, find $\omega \in \Omega$ which minimizes $\mathbb{E}_{\tau \sim P}\left[\mathcal{L}(\tau; \omega)\right]$.

If $\mathcal{L}$, Φ, and Ψ are differentiable, then gradient-based optimization algorithms, such as SGD and Adam (Kingma & Ba, 2015), can be used to solve the meta-solving problem. We call this approach gradient-based meta-solving (GBMS) as a generalization of gradient-based meta-learning as represented by MAML (Finn et al., 2017). As a typical example of the meta-solving problem, we consider learning how to choose good initial guesses of iterative solvers (Ajuria Illarramendi et al., 2020; Vaupel et al., 2020; Um et al., 2020; Özbay et al., 2021; Luna et al., 2021). Other examples are presented in Appendix A.

Example 1 (Generating initial guesses). Suppose that we need to repeatedly solve similar instances of a class of differential equations with a given iterative solver. The iterative solver requires an initial guess for each problem instance, which sensitively affects the accuracy and efficiency of the solution. Here, finding a strategy to optimally select an initial guess depending on each problem instance can be viewed as a meta-solving problem. For example, let us consider repeatedly solving 1D Poisson equations under different source terms. The task τ is to solve the 1D Poisson equation $-\frac{d^2}{dx^2}u(x) = f(x)$ with Dirichlet boundary condition $u(0) = u(1) = 0$. By discretizing it with the finite difference scheme, we obtain the linear system $Au = f$, where $A \in \mathbb{R}^{N \times N}$ and $u, f \in \mathbb{R}^N$. Thus, our task τ is represented by $\tau = \{f_\tau\}$, and the task distribution P is the distribution of f_τ. The loss function $\mathcal{L} : \mathcal{T} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ measures the accuracy of approximate solution $\hat{u} \in \mathcal{U} = \mathbb{R}^N$. For example, $\mathcal{L}(\tau, \hat{u}) = \|A\hat{u} - f_\tau\|^2$ is a possible choice. If we have a reference solution u_τ, then $\mathcal{L}(\tau, \hat{u}) = \|\hat{u} - u_\tau\|^2$ can be another choice. The solver $\Phi : \mathcal{T} \times \Theta \to \mathcal{U}$ is an iterative solver with an initial guess $\theta \in \Theta$ for the Poisson equation. For example, suppose that Φ is the Jacobi method. The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is a strategy characterized by $\omega \in \Omega$ to select the initial guess $\theta_\tau \in \Theta$ for each task $\tau \in \mathcal{T}$. For example, Ψ is a neural network with weight

ω. Then, finding the strategy to select initial guesses of the iterative solver becomes a meta-solving problem.

Besides scientific computing applications, the meta-solving framework also includes classical meta-learning problems, such as few-shot learning with MAML, where τ is a learning problem, Φ is one or few steps gradient descent for a neural network starting at initialization θ, Ψ is a constant function returning its weights ω, and $\omega = \theta$ is optimized to be easily fine-tuned for new tasks. We remark two key differences between Example 1 and the example of MAML. First, in Example 1, the initial guess θ_τ is selected for each task τ, while in MAML, the initialization θ is common to all tasks. Second, in the MAML example, we cannot take too many gradient descent steps in the inner loop at test time, because it can harm generalization. Thus, to be consistent with the test time setting, it is reasonable that we take a fixed small number of steps of gradient descent in the inner loop during training. By contrast, in Example 1, the inner loop is not a supervised learning but an iterative algorithm, and we can and often must take many iterations to obtain a solution of the required accuracy at test time.

Here, the second difference gives rise to the question about the choice of $\mathcal{L}$, which should be an indicator of how well the iterative solver Φ solves the task τ. However, if we iterate Φ until the solution error reaches a given tolerance ϵ, the error finally becomes ϵ and cannot be an indicator of solver performance. Instead, how fast the solver Φ finds a solution $\hat{u}$ satisfying the tolerance ϵ should be used as the performance indicator in this case. In fact, in most scientific applications, a required tolerance is set in advance, and iterative methods are assessed by the computational cost, in particular, the number of iterations, to achieve the tolerance (Saad, 2003). Thus, to be consistent with this test setting, the principled choice of loss function should be the number of iterations to reach a given tolerance ϵ, which we refer $\mathcal{L}_\epsilon$ through this paper.

2.2. Analysis of the Approach of Solution Error Minimization

Although $\mathcal{L}_\epsilon$ is the true loss function to be minimized, minimizing solution errors is the current main approach in the literature for learning parameters of iterative solvers. For example, in (Um et al., 2020), a neural network is used to generate an initial guess for the Conjugate Gradient (CG) solver. It is trained to minimize the solution error after a fixed number of CG iterations but evaluated by the number of CG iterations to reach a given tolerance. Similarly, (Chen et al., 2022) uses a neural network to generate the smoother of PDE-MgNet, a neural network representing the multigrid method. It is trained to minimize the solution error after one step of PDE-MgNet but evaluated by the number of steps of PDE-MgNet to reach a given tolerance.

These works can be interpreted that the solution error after m iterations, which we refer $\mathcal{L}_m$, is used as a surrogate of $\mathcal{L}_\epsilon$. Using the meta-solving framework, it can be understood that the current approach is trying to train meta-solver Ψ by applying gradient-based learning algorithms to the empirical version of

$$\min_{\omega} \mathop{\mathbb{E}}_{\tau \sim P}[\mathcal{L}_m(\tau, \Phi_m(\tau; \Psi(\tau; \omega)))] \tag{1}$$

as a surrogate of

$$\min_{\omega} \mathop{\mathbb{E}}_{\tau \sim P}[\mathcal{L}_\epsilon(\tau, \Phi_\epsilon(\tau; \Psi(\tau; \omega)))], \tag{2}$$

where Φ_m is an iterative solver whose stopping criterion is the maximum number of iterations m, and Φ_ϵ is the same kind of iterative solver but has a different stopping criterion, error tolerance ϵ.

The key question is now, is minimizing solution error $\mathcal{L}_m$ sufficient to minimize, at least approximately, the number of iterations $\mathcal{L}_\epsilon$? In other words, is (1) a valid surrogate for (2)? We hereafter show that the answer is negative. In fact, $\mathcal{L}_\epsilon$ can be arbitrarily large even if $\mathcal{L}_m$ is minimized, especially when the task difficulty (i.e. number of iterations required to achieve a fixed accuracy) varies widely between instances. This highlights a significant departure from classical meta-learning, and must be taken into account in algorithm design for scientific computing. Let us first illustrate this phenomenon using a numerical example and its concrete theoretical analysis. A resolution of this problem leading to our proposed method will be introduced in Section 3.

2.2.1. A Counter-example: Poisson Equation

Let us recall Example 1 and apply the current approach to it. The task $\tau = \{f_\tau\}$ is to solve the discretized 1D Poisson equation $Au = f_\tau$. During training, we use the Jacobi method Φ_m, which starts with the initial guess $\theta = \hat{u}^{(0)}$ and iterates m times to obtain the approximate solution $\hat{u}^{(m)} \in \mathcal{U}$. However, during testing, we use the Jacobi method Φ_ϵ whose stopping criterion is tolerance ϵ, which we set $\epsilon = 10^{-6}$ in this example. We consider two task distributions, P_1 and P_2, and their mixture $P = pP_1 + (1-p)P_2$ with weight $p \in [0, 1]$. P_1 and P_2 are designed to generate difficult (i.e. requiring a large number of iterations to solve) tasks and easy ones respectively (Appendix C.1). To generate the initial guess $\hat{u}^{(0)}$, two meta-solvers Ψ_{nn} and Ψ_{base} are considered. Ψ_{nn} is a fully-connected neural network with weights ω, which takes f_τ as inputs and generates $\hat{u}_\tau^{(0)}$ depending on each task τ. Ψ_{base} is a constant baseline, which gives the constant initial guess $\hat{u}^{(0)} = \mathbf{0}$ for all tasks. Note that Ψ_{base} is already a good choice, because $u_\tau(0) = u_\tau(1) = 0$ and $\mathbb{E}_{\tau\sim P}[u_\tau] = \mathbf{0}$. The loss function is the relative error after m iterations,

$\mathcal{L}_m = \left\|\hat{u}^{(m)} - A^{-1}f_\tau\right\|^2 / \left\|A^{-1}f_\tau\right\|^2$. Then, we solve (1) by GBMS as a surrogate of (2). We note that the case $m = 0$, where Φ_0 is the identity function that returns the initial guess, corresponds to using the solution predicted by ordinary supervised learning as the initial guess (Ajuria Illarramendi et al., 2020; Özbay et al., 2021). The case $m > 0$ is also studied in (Um et al., 2020). Other details, including the neural network architecture and hyper-parameters for training, are presented in Appendix C.1.

The trained meta-solvers are assessed by $\mathcal{L}_\epsilon$, the number of iterations to achieve the target tolerance $\epsilon = 10^{-6}$, which is written by $\mathcal{L}_\epsilon(\tau;\omega) = \min\{m \in \mathbb{Z}_{\geq 0} : \sqrt{L_m(\tau;\omega)} \leq \epsilon\}$ as a function of τ and ω.

Figure 1 compares the performance of the meta-solvers trained with different m, and it indicates the failure of the current approach (1). For the distribution of $p = 0$, where all tasks are sampled from the easy distribution P_2 and have similar difficulties, Ψ_{nn} successfully reduces the number of iterations by 76% compared to the baseline Ψ_{base} by choosing a good hyper-parameter $m = 25$. As for the hyper-parameter m, we note that larger m does not necessarily lead to better performance, which will be shown in Proposition 2.2, and the best training iteration number m is difficult to guess in advance. For the distribution of $p = 0.01$, where tasks are sampled from the difficult distribution P_1 with probability 0.01 and the easy distribution P_2 with probability 0.99, the performance of Ψ_{nn} degrades significantly. In this case, the reduction is only 26% compared to the baseline Ψ_{base} even for the tuned hyper-parameter.

Figure 2 illustrates the cause of this degradation. In the case of $p = 0$, $\mathcal{L}_m$ takes similar values for all tasks, and minimizing $\mathbb{E}_{\tau\sim P}[\mathcal{L}_m]$ can reduce the number of iterations for all tasks uniformly. However, in the case of $p = 0.01$, $\mathcal{L}_m$ takes large values for the difficult tasks and small values for the easy tasks, and $\mathbb{E}_{\tau\sim P}[\mathcal{L}_m]$ is dominated by a small number of difficult tasks. Consequently, the trained meta-solver with the loss $\mathcal{L}_m$ reduces the number of iterations for the difficult tasks but increases it for easy tasks consisting of the majority (Figure 2(e)). In other words, $\mathcal{L}_m$ is sensitive to difficult outliers, which degrades the performance of the meta-solver.

2.2.2. Analysis on the Counter-example

To understand the property of the current approach (1) and the cause of its performance degradation, we analyze the counter-example in a simpler setting. All of the proofs are presented in Appendix B.

Let our task space contain two tasks $\mathcal{T} = \{\tau_1, \tau_2\}$, where $\tau_i = \{f_i\}$ is a task to solve $Au = f_i$. To solve τ_i, we continue to use the Jacobi method Φ_m that starts with an initial guess $\hat{u}^{(0)}$ and gives an approximate solution $\hat{u} =$

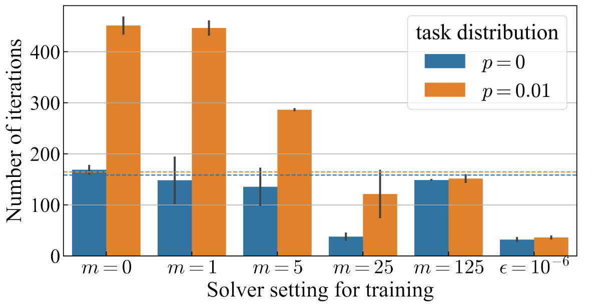


Figure 1. The average number of iterations of the Jacobi method to reach tolerance $\epsilon = 10^{-6}$ on the test data. The rightmost result is obtained by training with the proposed loss function $\tilde{\mathcal{L}}_\epsilon^{(m)}$, and the others are obtained by the conventional loss function $\mathcal{L}_m$. The two dotted lines indicate the baseline performance of Ψ_{base} for each task distribution of the corresponding color. The error bar indicates the standard deviation. Specific numbers are presented in Table 3.

$\hat{u}^{(m)}$. We assume that $f_i = c_i\mu_i v_i$, where $c_i \in \mathbb{R}_{\geq 0}$ is a constant, and v_i is the eigenvector of A corresponding to the eigenvalue μ_i. We also assume $\mu_1 < \mu_2$, which means τ_1 is more difficult than τ_2 for the Jacobi method. Let the task distribution P generates τ_1 with probability p and τ_2 with probability $1-p$ respectively. Let $\mathcal{L}_m(\tau_i, \hat{u}) = \left\|u^{(m)} - A^{-1}f_i\right\|^2$. We note that the analysis in this paper is also valid for a loss function measuring the residual, e.g. $\mathcal{L}_m(\tau_i, \hat{u}) = \left\|Au^{(m)} - f_i\right\|^2$. Suppose that the meta-solver Ψ gives a constant multiple of f as initial guess $\hat{u}^{(0)}$, i.e. $\hat{u}_i^{(0)} = \Psi(\tau_i;\omega) = \omega f_i$, where $\omega \in \mathbb{R}$ is the trainable parameter of Ψ.

For the above problem setting, we can show that the solution error minimization (1) has the unique minimizer $\omega_m = \operatorname{argmin}_\omega \mathbb{E}_{\tau\sim P}[\mathcal{L}_m(\tau, \Phi_m(\tau;\Psi(\tau;\omega)))]$ (Proposition B.2). As for the minimizer ω_m, we have the following result, which implies a problem in the current approach of solution error minimization.

Proposition 2.2. *For any $p \in (0,1)$, we have*

$$\lim_{m\to\infty} \omega_m = \frac{1}{\mu_1}. \tag{3}$$

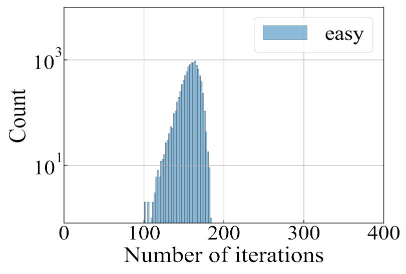
For any $\epsilon > 0$ and $p \in (0,1)$, there exists m_0 such that for any m_1 and m_2 satisfying $m_0 < m_1 < m_2$,

$$\mathop{\mathbb{E}}_{\tau\sim P}[\mathcal{L}_\epsilon(\tau;\omega_{m_1})] \leq \mathop{\mathbb{E}}_{\tau\sim P}[\mathcal{L}_\epsilon(\tau;\omega_{m_2})]. \tag{4}$$

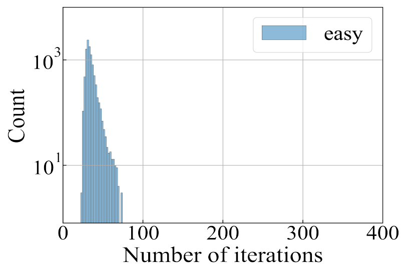
Furthermore, for any $\epsilon > 0$ and $M > 0$, there exists task space $(\mathcal{T}, P)$ such that for any $m \geq 0$,

$$\mathop{\mathbb{E}}_{\tau\sim P}[\mathcal{L}_\epsilon(\tau;\omega_m)] > M. \tag{5}$$

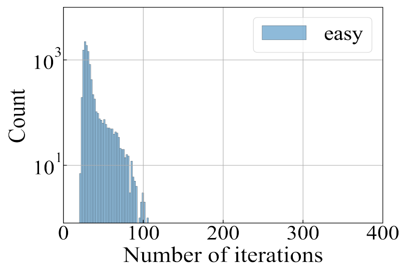
Note that if $\omega = 1/\mu_i$, the meta-solver Ψ gives the best initial guess, the solution, for τ_i, i.e. $\Psi(\tau_i; 1/\mu_i) = f_i/\mu_i =$



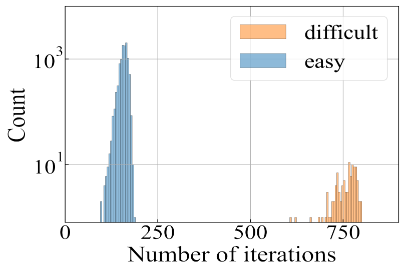
(a) $p = 0$ and Ψ_{base}



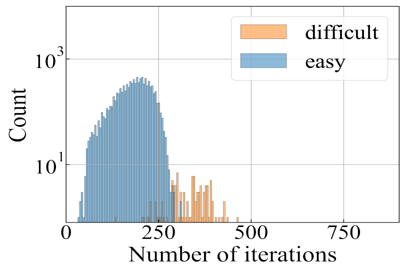
(b) $p = 0$ and Ψ_{nn} trained with $m = 25$



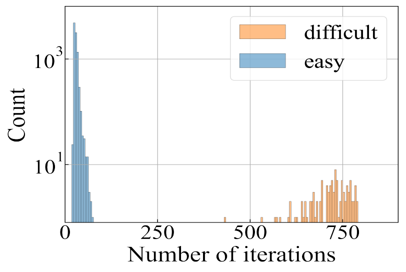
(c) $p = 0$ and Ψ_{nn} trained with $\epsilon = 10^{-6}$



(d) $p = 0.01$ and Ψ_{base}



(e) $p = 0.01$ and Ψ_{nn} trained with $m = 25$



(f) $p = 0.01$ and Ψ_{nn} trained with $\epsilon = 10^{-6}$

Figure 2. Comparison of distributions of the number of iterations to reach tolerance $\epsilon = 10^{-6}$ on the test data.

$A^{-1}f_i$. Proposition 2.2 shows the solution error is not a good surrogate loss for learning to accelerate iterative methods in the following sense. First, Equation (3) shows that $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_m]$ is dominated by the loss for τ_1 as m becomes large. Hence, τ_2 is ignored regardless of its probability $1-p$, which can cause a problem when $1-p$ is large. Second, Inequality (4) implies that training with more iterative steps may lead to worse performance. This is because with larger m, $\mathcal{L}_m$ is more dominated by the difficult task τ_1, and ω_m goes closer to $1/\mu_1$. However, the optimal ω minimizing $\mathcal{L}_\epsilon$, which will be explained in Proposition 3.1, lies in the interval $(1/\mu_2, 1/\mu_1)$. Hence, if we increase m, ω_m moves away from the optimal ω and leads to worse performance. This performance degradation can happen regardless of p, but when p is small, it is more critical because the optimal ω is near $1/\mu_2$. Finally, Inequality (5) shows that the end result can be arbitrarily bad. These are consistent with numerical results in Section 2.2.1. This motivates our subsequent proposal of a principled approach to achieve acceleration of iterative methods using meta-learning techniques.

3. A Novel Approach to Accelerate Iterative Methods

As we observe from the example presented in Section 2, the main issue is that in general, the solution error $\mathcal{L}_m$ is not a valid surrogate for the number of iterations $\mathcal{L}_\epsilon$. Hence, the resolution of this problem amounts to finding a valid surrogate for $\mathcal{L}_\epsilon$ that is also amenable to training. An advantage of our formalism for GBMS introduced in Section 2.1 is that it tells us exactly the right loss to minimize, which is precisely (2). It remains then to find an approximation of this loss function, and this is the basis of the proposed method, which will be explained in Section 3.2. First, we return to the example in Section 2.2.2 and show now that minimizing $\mathcal{L}_\epsilon$ instead of $\mathcal{L}_m$ guarantees performance improvement.

3.1. Minimizing the Number of Iterations

3.1.1. Analysis on the Counter-example

To see the property and advantage of directly minimizing $\mathcal{L}_\epsilon$, we analyze the same example in Section 2.2.2. The problem setting is the same as in Section 2.2.2, except that we minimize $\mathcal{L}_\epsilon$ instead of $\mathcal{L}_m$ with Φ_ϵ instead of Φ_m. Note that we relax the number of iterations $m \in \mathbb{Z}_{\geq 0}$ to $m \in \mathbb{R}_{\geq 0}$ for the convenience of the analysis. All of the proofs are presented in Appendix B.

As in Section 2.2.2, we can find the minimizer $\omega_\epsilon = \operatorname{argmin}_\omega \mathbb{E}_{\tau \sim P}[\mathcal{L}_\epsilon(\tau, \Phi_\epsilon(\tau; \Psi(\tau; \omega)))]$ by simple calculation (Proposition B.3). As for the minimizer ω_ϵ, we have the counterpart of Proposition 2.2:

Proposition 3.1. *We have*

$$\lim_{\epsilon \to 0} \omega_\epsilon = \begin{cases} \frac{1}{\mu_1} & \text{if } p > p_0 \\ \frac{1}{\mu_2} & \text{if } p < p_0, \end{cases} \tag{6}$$

where p_0 is a constant depending on μ_1 and μ_2. If $\frac{c_1 c_2(\mu_2 - \mu_1)}{2(c_1\mu_1 + c_2\mu_2)} > \delta_1 > \delta_2$ and $p \notin (p_0, p_{\delta_1})$, then for any $\epsilon \leq \delta_2$,

$$\mathop{\mathbb{E}}_{\tau \sim P}[\mathcal{L}_\epsilon(\tau; \omega_{\delta_1})] > \mathop{\mathbb{E}}_{\tau \sim P}[\mathcal{L}_\epsilon(\tau; \omega_{\delta_2})], \tag{7}$$

where p_δ is a constant depending on tolerance δ expained in Proposition B.3, and p_0 is its limit as δ tends to 0. Furthermore, for any $c > 0$, $\epsilon > 0$, and $\delta \geq \epsilon$, there exists a task space $(\mathcal{T}, P)$ such that for any $m \geq 0$,

$$\mathop{\mathbb{E}}_{\tau\sim P}[\mathcal{L}_\epsilon(\tau;\omega_m)] > c \mathop{\mathbb{E}}_{\tau\sim P}[\mathcal{L}_\epsilon(\tau;\omega_\delta)]. \tag{8}$$

Note that the assumption $\delta_1 < \frac{c_1c_2(\mu_2-\mu_1)}{2(c_1\mu_1+c_2\mu_2)}$ is to avoid the trivial case, $\min_\omega \mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)] = 0$, and the interval (p_0, p_{δ_1}) is reasonably narrow for such small δ_1. From Proposition 3.1, the following insights can be gleaned. First, the minimizer ω_ϵ has different limits depending on task probability and difficulty, while the limit of ω_m is always determined by the difficult task. In other words, $\mathcal{L}_\epsilon$ and $\mathcal{L}_m$ weight tasks differently. Second, Inequality (7) guarantees the improvement, that is, training with the smaller tolerance δ_2 leads to better performance for a target tolerance $\epsilon \leq \delta_2$. We remark that, as shown in Proposition 2.2, this kind of guarantee is not available in the current approach, where a larger number of training iterations m_2, i.e. larger training cost, does not necessarily improve the performance. Furthermore, (8) implies the existence of a problem where the current method, even with hyper-parameter tuning, performs arbitrarily worse than the right problem formulation (2). These clearly show the advantage and necessity of a valid surrogate loss function for $\mathcal{L}_\epsilon$, which we will introduce in Section 3.2.

3.1.2. Resolving the Counter-example

Before introducing our surrogate loss function, we present numerical evidence that the counter-example, solving Poisson equations in Section 2.2.1, is resolved by directly minimizing the number of iterations $\mathcal{L}_\epsilon$. The problem setting is the same as in Section 2.2.1 except for the solver and the loss function. Instead of solver Φ_m, we use solver Φ_ϵ that stops when the error $\mathcal{L}_m$ becomes smaller than ϵ. Also, we directly minimize $\mathcal{L}_\epsilon$ instead of $\mathcal{L}_m$. Although $\mathcal{L}_\epsilon$ is discrete and (2) cannot be solved by gradient-based algorithms, we can use a differentiable surrogate loss function $\tilde{\mathcal{L}}_\epsilon$ for $\mathcal{L}_\epsilon$, which will be explained in Section 3.2. The following numerical result is obtained by solving the surrogate problem (11) by Algorithm 1.

Figure 1 and Figure 2 show the advantage of the proposed approach (2) over the current one (1). For the case of $p = 0$, although the current method can perform comparably with ours by choosing a good hyper-parameter $m = 25$, ours performs better without the hyper-parameter tuning. For the case of $p = 0.01$, the current method degrades and performs poorly even with the hyper-parameter tuning, while ours keeps its performance and reduces the number of iterations by 78% compared to the constant baseline Ψ_{base} and 70% compared to the best-tuned current method ($m = 25$). We note that this is the case implied in Proposition 3.1. Figure 2 illustrates the cause of this performance difference. While the current method is distracted by a few difficult tasks and increases the number of iterations for the majority of easy tasks, our method reduces the number of iterations for all tasks in particular for the easy ones. This difference originates from the property of $\mathcal{L}_m$ and $\mathcal{L}_\epsilon$ explained in Section 3.1.1. That is, $\mathcal{L}_m$ can be dominated by the small number of difficult tasks whereas $\mathcal{L}_\epsilon$ can balance tasks of different difficulties depending on their difficulty and probability.

3.2. Surrogate Problem

We have theoretically and numerically shown the advantage of directly minimizing the number of iterations $\mathcal{L}_\epsilon$ over the current approach of minimizing $\mathcal{L}_m$. Now, the last remaining issue is how to minimize $\mathbb{E}_{\tau\sim P}[\mathcal{L}_\epsilon]$. As with any machine learning problem, we only have access to finite samples from the task space $(\mathcal{T}, P)$. Thus, we want to minimize the empirical loss: $\min_\omega \frac{1}{N}\sum_{i=1}^N \mathcal{L}_\epsilon(\tau_i;\omega)$. However, the problem is that $\mathcal{L}_\epsilon$ is not differentiable with respect to ω. To overcome this issue, we introduce a differentiable surrogate loss function $\tilde{\mathcal{L}}_\epsilon$ for $\mathcal{L}_\epsilon$.

To design the surrogate loss function, we first express $\mathcal{L}_\epsilon$ by explicitly counting the number of iterations. We define $\mathcal{L}_\epsilon^{(m)}$ by

$$\mathcal{L}_\epsilon^{(k+1)}(\tau;\omega) = \mathcal{L}_\epsilon^{(k)}(\tau;\omega) + \mathbb{1}_{\mathcal{L}_k(\tau;\omega)>\epsilon}, \tag{9}$$

where $\mathcal{L}_\epsilon^{(0)}(\tau;\omega) = 0$, $\mathcal{L}_k$ is any differentiable loss function that measures the quality of the k-th step solution $\hat{u}^{(k)}$, and $\mathbb{1}_{\mathcal{L}_k(\tau;\omega)>\epsilon}$ is the indicator function that returns 1 if $\mathcal{L}_k(\tau;\omega) > \epsilon$ and 0 otherwise. Then, we have $\mathcal{L}_\epsilon(\tau;\omega) = \lim_{m\to\infty}\mathcal{L}_\epsilon^{(m)}(\tau;\omega)$ for each τ and ω. Here, $\mathcal{L}_\epsilon^{(m)}$ is still not differentiable because of the indicator function. Thus, we define $\tilde{\mathcal{L}}_\epsilon^{(m)}$ as a surrogate for $\mathcal{L}_\epsilon^{(m)}$ by replacing the indicator function with the sigmoid function with gain parameter a (Yin et al., 2019). In summary, $\tilde{\mathcal{L}}_\epsilon^{(m)}$ is defined by

$$\tilde{\mathcal{L}}_\epsilon^{(k+1)}(\tau;\omega) = \tilde{\mathcal{L}}_\epsilon^{(k)}(\tau;\omega) + \sigma_a(\mathcal{L}_k(\tau;\omega) - \epsilon). \tag{10}$$

Then, we have $\mathcal{L}_\epsilon(\tau;\omega) = \lim_{m\to\infty}\lim_{a\to\infty}\tilde{\mathcal{L}}_\epsilon^{(m)}(\tau;\omega)$ for each τ and ω.

However, in practice, the meta-solver Ψ may generate a bad solver parameter, particularly in the early stage of training. The bad parameter can make $\mathcal{L}_\epsilon$ very large or infinity at the worst case, and it can slow down or even stop the training. To avoid this, we fix the maximum number of iterations m sufficiently large and use $\tilde{\mathcal{L}}_\epsilon^{(m)}$ as a surrogate for $\mathcal{L}_\epsilon$ along with solver $\Phi_{\epsilon,m}$, which stops when the error reaches tolerance ϵ or the number of iterations reaches m. Note that $\Phi_\epsilon = \Phi_{\epsilon,\infty}$, $\Phi_m = \Phi_{0,m}$, but $\Phi_m \neq \Phi_{\epsilon,m}$ in this notation. Φ_m always do m iterations, but $\Phi_{\epsilon,m}$ can stop before m

iterations when the error reaches tolerance ϵ. In summary, a surrogate problem for (2) is defined by

$$\min_{\omega} \mathop{\mathbb{E}}_{\tau\sim P}[\tilde{\mathcal{L}}_{\epsilon}^{(m)}(\tau, \Phi_{\epsilon,m}(\tau; \Psi(\tau;\omega)))], \tag{11}$$

which can be solved by GBMS (Algorithm 1).

Algorithm 1 GBMS for minimizing the number of iterations

```
  Input: $(P,\mathcal{T})$: task space, $\Psi$: meta-solver, $\Phi$: iterative solver, $\phi$: iterative function of $\Phi$, $m$: maximum number of iterations of $\Phi$, $\epsilon$: tolerance of $\Phi$, $\mathcal{L}$: loss function for solution quality, $S$: stopping criterion for outer loop, Opt: gradient-based algorithm
  while $S$ is not satisfied do
    $\tau \sim P$                                              ▷sample task $\tau$ from $P$
    $\theta_\tau \leftarrow \Psi(\tau;\omega)$                 ▷generate $\theta_\tau$ by $\Psi$ with $\omega$
    $k \leftarrow 0$, $\hat{u} \leftarrow \Phi_0(\tau;\theta_\tau)$, $\tilde{\mathcal{L}}_\epsilon(\tau,\hat{u}) \leftarrow 0$    ▷initialize
    while $k < m$ or $\mathcal{L}(\tau,\hat{u}) > \epsilon$ do
      $k \leftarrow k+1$, $\tilde{\mathcal{L}}_\epsilon(\tau,\hat{u}) \leftarrow \tilde{\mathcal{L}}_\epsilon(\tau,\hat{u}) + \sigma_a(\mathcal{L}(\tau,\hat{u}) - \epsilon)$
      ▷count iterations and compute surrogate loss
      $\hat{u} \leftarrow \phi(\hat{u};\theta_\tau)$           ▷iterate $\phi$ with $\theta_\tau$ to update $\hat{u}$
    end while
    $\omega \leftarrow \mathrm{Opt}(\omega, \nabla_\omega \tilde{\mathcal{L}}_\epsilon(\tau,\hat{u}))$   ▷update $\omega$ to minimize $\tilde{\mathcal{L}}_\epsilon$
  end while
```

4. Numerical Examples

In this section, we show high-performance and versatility of the proposed method (Algorithm 1) using numerical examples in more complex scenarios, involving different task spaces, different iterative solvers and their parameters. Only the main results are presented in this section; details are provided in Appendix C.

4.1. Generating Relaxation Factors

This section presents the application of the proposed method for generating relaxation factors of the SOR method. The SOR method is an improved version of the Gauss-Seidel method with relaxation factors enabling faster convergence (Saad, 2003). Its performance is sensitive to the choice of the relaxation factor, but the optimal choice is only accessible in simple cases (e.g. (Yang & Gobbert, 2009)) and is difficult to obtain beforehand in general. Thus, we aim to choose a good relaxation factor to accelerate the convergence of the SOR method using Algorithm 1. Moreover, leveraging the generality of Algorithm 1, we generate the relaxation factor and initial guess simultaneously and observe its synergy.

We consider solving Robertson equation (Robertson, 1966), which is a nonlinear ordinary differential equation that models a certain reaction of three chemicals in the form:

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} -c_1y_1 + c_3y_2y_3 \\ c_1y_1 - c_2y_2^2 - c_3y_2y_3 \\ c_2y_2^2 \end{pmatrix}, \tag{12}$$

$$\begin{pmatrix} y_1(0) \\ y_2(0) \\ y_3(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad t \in [0,T], \tag{13}$$

We write $(y_1, y_2, y_3)^T$ as y and the right hand side of Equation (12) as $f(y)$. Since the Robertson equation is stiff, it is solved by implicit numerical methods (Hairer & Wanner, 2010). For simplicity, we use the backward Euler method: $y_{n+1} = y_n + h_nf(y_{n+1})$, where h_n is the step size. To find y_{n+1}, we need to solve the nonlinear algebraic equation

$$g_n(y_{n+1}) := y_{n+1} - h_nf(y_{n+1}) - y_n = 0 \tag{14}$$

at every time step. For solving (14), we use the one-step Newton-SOR method (Saad, 2003). Our task τ is to solve (14) using the Newton-SOR method, which is represented by $\tau = \{c_1, c_2, c_3, h_n, y_n\}$. Note that τ does not contain the solution of (14), so the training is done in an unsupervised manner. The loss function $\mathcal{L}_m$ is the residual $\left\|g_n(y_{n+1}^{(m)})\right\|$, and $\mathcal{L}_\epsilon$ is the number of iterations to have $\mathcal{L}_m < \epsilon$. As for the task distribution, we sample c_1, c_2, c_3 log-uniformly from $[10^{-4}, 1]$, $[10^5, 10^9]$, $[10^2, 10^6]$ respectively. The solver $\Phi_{\epsilon,m}$ is the Newton-SOR method and its parameter θ is a pair of relaxation factor and initial guess $(r, y_{n+1}^{(0)})$. At test time, we set $m = 10^4$ and $\epsilon = 10^{-9}$. We compare four meta-solvers Ψ_{base}, Ψ_{ini}, Ψ_{relax}, and Ψ_{both}. The meta-solver Ψ_{base} has no parameters, while Ψ_{ini}, Ψ_{relax}, and Ψ_{both} are implemented by fully-connected neural networks with weight ω. The meta-solver Ψ_{base} is a classical baseline, which uses the previous time step solution y_n as an initial guess $y_{n+1}^{(0)}$ and a constant relaxation factor r_{base}. The constant relaxation factor is $r_{\text{base}} = 1.37$, which does not depend on task τ and is chosen by the brute force search to minimize the average number of iterations to reach target tolerance $\epsilon = 10^{-9}$ in the whole training data. The meta-solver Ψ_{ini} generates an initial guess y_τ adaptively, but uses the constant relaxation factor r_{base}. The meta-solver Ψ_{relax} generates a relaxation factor $r_\tau \in [1, 2]$ adaptively but uses the previous time step solution as an initial guess. The meta-solver Ψ_{both} generates both adaptively. Then, the meta-solvers are trained by GBMS with formulations (1) and (11) depending on the solver $\Phi_{\epsilon,m}$.

The results are presented in Table 1. The significant advantage of the proposed approach (11) over the baselines and the current approach (1) is observed in Table 1(a). The best meta-solver Ψ_{both} with $\Phi_{10^{-9},2000}$ reduces the number of iterations by 87% compared to the baseline, while all the meta-solvers trained in the current approach (1) fail to outperform the baseline. This is because the relaxation factor that minimizes the number of iterations for a given tolerance can be very different from the relaxation factor that minimizes the residual at a given number of iterations. Furthermore, Table 1(b) implies that simultaneously generating the initial guess and relaxation factor can create synergy. This follows from the observation that the degree of improvement by Ψ_{both} from Ψ_{ini} and Ψ_{relax} (83% and 29%) are greater than that by Ψ_{relax} and Ψ_{ini} from Ψ_{base} (80% and 20%).

Table 1. The average number of iterations $\mathcal{L}_\epsilon$ of the Newton-SOR method to reach tolerance $\epsilon = 10^{-9}$ on the test data.

(a) Different training settings.

Ψ	$\mathcal{L}$	ϵ	m	$\mathcal{L}_\epsilon$
Ψ_{base}	-	-	-	70.51
Ψ_{both}	$\mathcal{L}_m$	-	1	199.20
		-	5	105.71
		-	25	114.43
		-	125	109.24
		-	625	190.56
	$\tilde{\mathcal{L}}_\epsilon^{(m)}$	10^{-9}	2000	**9.51**

(b) Different meta-solvers.

Ψ	$\mathcal{L}$	ϵ	m	$\mathcal{L}_\epsilon$
Ψ_{base}	-	-	-	70.51
Ψ_{ini}	$\tilde{\mathcal{L}}_\epsilon^{(m)}$	10^{-9}	2000	55.82
Ψ_{relax}				14.38
Ψ_{both}				**9.51**

4.2. Generating Preconditioning Matrices

In this section, Algorithm 1 is applied to preconditioning of the Conjugate Gradient (CG) method. The CG method is an iterative method that is widely used in solving large sparse linear systems, especially those with symmetric positive definite matrices. Its performance is dependent on the condition number of the coefficient matrix, and a variety of preconditioning methods have been proposed to improve it. In this section, we consider ICCG(α), which is the CG method with incomplete Cholesky factorization (ICF) preconditioning with diagonal shift α. In ICCG(α), the preconditioning matrix $M = \tilde{L}\tilde{L}^T$ is obtained by applying ICF to $A + \alpha I$ instead of A. If α is small, $A + \alpha I$ is close to A but the decomposition $\tilde{L}\tilde{L}^T \approx A + \alpha I$ may have poor quality or even fail. If α is large, the decomposition $\tilde{L}\tilde{L}^T \approx A + \alpha I$ may have good quality but $A + \alpha I$ is far from the original matrix A. Although α can affect the performance of ICCG(α), there is no well-founded solution to how to choose α for a given A (Saad, 2003). Thus, we train a meta-solver to generate α depending on each problem instance.

We consider linear elasticity equations:

$$-\nabla \cdot \sigma(u) = f \quad \text{in } B \tag{15}$$

$$\sigma(u) = \lambda \operatorname{tr}(\epsilon(u))I + 2\mu\epsilon(u) \tag{16}$$

$$\epsilon(u) = \frac{1}{2}\left(\nabla u + (\nabla u)^T\right) \tag{17}$$

where u is the displacement field, σ is the stress tensor, ϵ is strain-rate tensor, λ and μ are the Lamé elasticity parameters, and f is the body force. Specifically, we consider clamped beam deformation under its own weight. The beam is clamped at the left end, i.e. $u = (0, 0, 0)^T$ at $x = 0$, and has length 1 and a square cross-section of width W. The force f is gravity acting on the beam, i.e. $f = (0, 0, -\rho g)^T$, where ρ is the density and g is the acceleration due to gravity. The model is discretized using the finite element method with $8 \times 8 \times 2$ cuboids mesh, and the resulting linear system $Au = f$ has $N = 243$ unknowns. Our task τ is to solve the linear system $Au = f$, and it is represented by $\tau = \{W_\tau, \lambda_\tau, \mu_\tau, \rho_\tau\}$. Each pysical parameter is sampled from $W_\tau \sim \text{LogUniform}(0.003, 0.3)$, $\lambda_\tau, \mu_\tau, \rho_\tau \sim \text{LogUniform}(0.1, 10)$, which determine the task space $(\mathcal{T}, P)$. Our solver $\Phi_{\epsilon,m}$ is the ICCG method and its parameter is diagonal shift α. At test time, we set $m = N = 243$, the coefficient matrix size, and $\epsilon = 10^{-6}$. We compare three meta-solvers $\Psi_{\text{base}}, \Psi_{\text{best}}$ and Ψ_{nn}. The meta-solver Ψ_{base} is a baseline that provides a constant $\alpha = 0.036$, which is the minimum α that succeeds ICF for all training tasks. The meta-solver Ψ_{best} is another baseline that generates the optimal α_τ for each task τ by the brute force search. The meta-solver Ψ_{nn} is a fully-connected neural network that takes τ as inputs and outputs α_τ for each τ. The loss function $\mathcal{L}_m$ is the relative residual $\mathcal{L}_m = \left\|A\hat{u}^{(m)} - f\right\| / \|f\|$, and the $\mathcal{L}_\epsilon$ is the number of iterations to achieve target tolerance $\epsilon = 10^{-6}$. In addition, we consider a regularization term $\mathcal{L}_{\text{ICF}}(\tau; \omega) = -\log \alpha_\tau$ to penalize the failure of ICF. By combining them, we use $\mathbb{1}_{\text{success}}\mathcal{L}_m + \gamma \mathbb{1}_{\text{fail}}\mathcal{L}_{\text{ICF}}$ for training with Φ_m and $\mathbb{1}_{\text{success}}\tilde{\mathcal{L}}_\epsilon + \gamma \mathbb{1}_{\text{fail}}\mathcal{L}_{\text{ICF}}$ for training with Φ_ϵ, where γ is a hyperparameter that controls the penalty for the failure of ICF, and $\mathbb{1}_{\text{success}}$ and $\mathbb{1}_{\text{fail}}$ indicates the success and failure of ICF respectively. Then, the meta-solvers are trained by GBMS with formulations (1) and (11) depending on the solver $\Phi_{\epsilon,m}$.

Table 2 shows the number of iterations of ICCG with diagonal shift α generated by the trained meta-solvers. For this problem setting, only one meta-solver trained by the current approach with the best hyper-parameter ($m = 125$) outperforms the baseline. All the other meta-solvers trained by the current approach increase the number of iterations and even fail to converge for the majority of problem instances. This may be because the error of m-th step solution $\mathcal{L}_m$ is less useful for the CG method. Residuals of the CG method do not decrease monotonically, but often remain high and oscillate for a while, then decrease sharply. Thus, $\mathcal{L}_m$ is not a good indicator of the performance of the CG method. This property of the CG method makes it more difficult to learn good α for the current approach. On the other hand, the meta-solver trained by our approach converges for most instances and reduces the number of iterations by 55% compared to Ψ_{base} and 33% compared to the best-tuned current approach ($m = 125$) without the hyper-parameter tuning. Furthermore, the performance of the proposed approach is close to the optimal choice Ψ_{best}.

Table 2. The average number of iterations $\mathcal{L}_\epsilon$ of ICCG to reach tolerance $\epsilon = 10^{-6}$ on the test data. If ICF fails, it is counted as the maximum number of iterations $N = 243$.

Ψ	$\mathcal{L}$	ϵ	m	$\mathcal{L}_\epsilon$	convergence ratio
Ψ_{base}	-	-	-	115.26	0.864
Ψ_{best}	-	-	-	44.53	1.00
Ψ_{nn}	$\mathcal{L}_m$	-	1	204.93	0.394
		-	5	183.90	0.626
		-	25	193.80	0.410
		-	125	77.08	**1.00**
		-	243	201.65	0.420
	$\tilde{\mathcal{L}}_\epsilon^{(m)}$	10^{-6}	243	**52.02**	**0.999**

5. Discussion

Other objectives In this paper, we compared two objectives $\mathcal{L}_m$ and $\mathcal{L}_\epsilon$, but there are other possible objectives, such as the condition number of the preconditioned matrix (Calì et al., 2023), the spectral radius of the iteration matrix (Luz et al., 2020), and the coarse grid complexity of the multigrid method (Taghibakhshi et al., 2021). They are similar to ours in the sense that they try to minimize the computation cost rather than the solution error. However, these objectives are often problem-specific and require some knowledge about the target problem and algorithm. On the other hand, our method can be generally applied to iterative algorithms. Moreover, there are avenues to combine these methods to further improve performance.

Limitations A limitation of the proposed method is that Algorithm 1 requires the differentiability of solver Φ. To avoid this limitation, there are two possible approaches. The first one is modifying the target solver to be differentiable. In fact, differentiable variants of traditional solvers are actively studied recently to combine them with neural networks (Hsieh et al., 2018; Chen et al., 2022). These differentiable solvers are usually implemented in deep learning frameworks and can be used as a component of our method out-of-the-box. However, currently, many real application problems are solved using well-tested existing solvers that do not support automatic differentiation. For such legacy solvers, we can resort to the second approach, where we use the finite difference method instead of backpropagation to compute $\partial\Phi/\partial\theta$ and then connect it to $\partial\Psi/\partial\omega$ computed by backpropagation. A similar idea is found in implementations of traditional numerical methods (e.g. optimization methods in SciPy (Virtanen et al., 2020)), where finite difference methods are implemented to handle black-box functions whose derivatives are not provided.

6. Conclusion

In this paper, we proposed a formulation of meta-solving as a general framework to analyze and develop learning-based iterative numerical methods. Under the framework, we identify limitations of current approaches directly based on meta-learning, and make concrete the mechanisms specific to scientific computing resulting in its failure. This is supported by both numerical and theoretical arguments. The understanding then leads to a simple but novel training approach that alleviates this issue. In particular, we proposed a practical surrogate loss function for directly minimizing the expected computational overhead, and demonstrated the high-performance and versatility of the proposed method through a range of numerical examples. Our analysis highlights the importance of precise formulations and the necessity of modifications when adapting data-driven workflows to scientific computing.

In future work, more theoretical analysis of the proposed method is needed. For example, the essence of the counterexample may hold for more general settings, and the property of the proposed surrogate loss (11) is worth studying in more detail. Besides the theoretical analysis, application research is another direction. For instance, our method can be extended for black-box legacy solvers to solve real engineering problems as discussed in Section 5. Moreover, in addition to traditional numerical algorithms, our approach can be used for learning to optimize neural networks for faster training.

Acknowledgement

We would like to thank the anonymous reviewers for their constructive comments. S. Arisaka is supported by Kajima Corporation, Japan. Q. Li is supported by the National Research Foundation, Singapore, under the NRF fellowship (NRF-NRFF13-2021-0005).

References

Ajuria Illarramendi, E., Alguacil, A., Bauerheim, M., Misdariis, A., Cuenot, B., and Benazera, E. Towards an hybrid computational strategy based on Deep Learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*, AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, June 2020. doi: 10.2514/6.2020-3058. URL https://doi.org/10.2514/6.2020-3058.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. The FEniCS Project Version 1.5. *Anschnitt*, 3(100), December 2015. ISSN 0003-5238. doi: 10.11588/ans.2015.100.20553. URL https://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553.

Azulay, Y. and Treister, E. Multigrid-Augmented Deep Learning Preconditioners for the Helmholtz Equation. *SIAM Journal of Scientific Computing*, pp. S127–S151, August 2022. ISSN 1064-8275. doi: 10.1137/21M1433514. URL https://doi.org/10.1137/21M1433514.

Calì, S., Hackett, D. C., Lin, Y., Shanahan, P. E., and Xiao, B. Neural-network preconditioners for solving the Dirac equation in lattice gauge theory. *Physical Review D*, 107 (3):034508, February 2023. doi: 10.1103/PhysRevD.107.034508. URL https://link.aps.org/doi/10.1103/PhysRevD.107.034508.

Chen, Y., Dong, B., and Xu, J. Meta-MgNet: Meta multigrid networks for solving parameterized partial differential equations. *Journal of computational physics*, 455:110996, April 2022. ISSN 0021-9991. doi: 10.1016/j.jcp.2022.110996. URL https://www.sciencedirect.com/science/article/pii/S0021999122000584.

Cheng, L., Illarramendi, E. A., Bogopolsky, G., Bauerheim, M., and Cuenot, B. Using neural networks to solve the 2D Poisson equation for electric field computation in plasma fluid simulations. September 2021. URL http://arxiv.org/abs/2109.13076.

Chou, I.-C. and Voit, E. O. Recent developments in parameter estimation and structure identification of biochemical and genomic systems. *Mathematical biosciences*, 219 (2):57–83, June 2009. ISSN 0025-5564, 1879-3134. doi: 10.1016/j.mbs.2009.03.002. URL http://dx.doi.org/10.1016/j.mbs.2009.03.002.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks: the official journal of the International Neural Network Society*, 107: 3–11, November 2018. ISSN 0893-6080, 1879-2782. doi: 10.1016/j.neunet.2017.12.012. URL http://dx.doi.org/10.1016/j.neunet.2017.12.012.

Feliu-Fabà, J., Fan, Y., and Ying, L. Meta-learning pseudo-differential operators with deep neural networks. *Journal of computational physics*, 408:109309, May 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109309. URL https://www.sciencedirect.com/science/article/pii/S0021999120300838.

Finn, C., Abbeel, P., and Levine, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Gallet, A., Rigby, S., Tallman, T. N., Kong, X., Hajirasouliha, I., Liew, A., Liu, D., Chen, L., Hauptmann, A., and Smyl, D. Structural engineering from an inverse problems perspective. *Proceedings. Mathematical, physical, and engineering sciences / the Royal Society*, 478(2257):20210526, January 2022. ISSN 1364-5021. doi: 10.1098/rspa.2021.0526. URL http://dx.doi.org/10.1098/rspa.2021.0526.

Greenbaum, A. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, January 1997. ISBN 9780898713961. doi: 10.1137/1.9781611970937. URL https://doi.org/10.1137/1.9781611970937.

Guo, X., Li, W., and Iorio, F. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 481–490, New York, NY, USA, August 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939738. URL https://doi.org/10.1145/2939672.2939738.

Guo, Y., Dietrich, F., Bertalan, T., Doncevic, D. T., Dahmen, M., Kevrekidis, I. G., and Li, Q. Personalized Algorithm Generation: A Case Study in Learning ODE Integrators. *SIAM Journal of Scientific Computing*, 44 (4):A1911–A1933, August 2022. ISSN 1064-8275. doi: 10.1137/21M1418629. URL https://doi.org/10.1137/21M1418629.

Hairer, E. and Wanner, G. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Berlin Heidelberg, March 2010. ISBN 9783642052200. doi: 10.1007/978-3-642-05221-

7. URL https://play.google.com/store/books/details?id=em9aAQAACAAJ.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. Meta-Learning in Neural Networks: A Survey. *IEEE transactions on pattern analysis and machine intelligence*, PP, May 2021. ISSN 0162-8828. doi: 10.1109/TPAMI.2021.3079209. URL http://dx.doi.org/10.1109/TPAMI.2021.3079209.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning Neural PDE Solvers with Convergence Guarantees. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/pdf?id=rklaWn0qK7.

Huang, J., Wang, H., and Yang, H. Int-Deep: A deep learning initialized iterative method for nonlinear problems. *Journal of computational physics*, 419:109675, October 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109675. URL https://www.sciencedirect.com/science/article/pii/S0021999120304496.

Huang, T., Zhang, Y., Li, L., Shao, W., and Lai, S.-J. Modified incomplete Cholesky factorization for solving electromagnetic scattering problems. *Progress in electromagnetics research B. Pier B*, 13:41–58, 2009. URL https://www.jpier.org/PIERB/pierb13/03.08112407.pdf.

Huang, X., Ye, Z., Liu, H., Ji, S., Wang, Z., Yang, K., Li, Y., Wang, M., Chu, H., Yu, F., Hua, B., Chen, L., and Dong, B. Meta-Auto-Decoder for Solving Parametric Partial Differential Equations. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 23426–23438. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/948552777302d3abf92415b1d7e9de70-Paper-Conference.pdf.

Kaneda, A., Akar, O., Chen, J., Kala, V., Hyde, D., and Teran, J. A Deep Conjugate Direction Method for Iteratively Solving Linear Systems. May 2022. URL http://arxiv.org/abs/2205.10763.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Liu, X., Zhang, X., Peng, W., Zhou, W., and Yao, W. A novel meta-learning initialization method for physics-informed neural networks. *Neural computing & applications*, 34(17):14511–14534, September 2022. ISSN 0941-0643, 1433-3058. doi: 10.1007/s00521-022-07294-2. URL https://doi.org/10.1007/s00521-022-07294-2.

Luna, K., Klymko, K., and Blaschke, J. P. Accelerating GMRES with Deep Learning in Real-Time. March 2021. URL http://arxiv.org/abs/2103.10975.

Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I. Learning Algebraic Multigrid Using Graph Neural Networks. In Iii, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6489–6499. PMLR, 2020. URL https://proceedings.mlr.press/v119/luz20a.html.

Moles, C. G., Mendes, P., and Banga, J. R. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome research*, 13(11): 2467–2474, November 2003. ISSN 1088-9051. doi: 10.1101/gr.1262503. URL http://dx.doi.org/10.1101/gr.1262503.

Nikolopoulos, S., Kalogeris, I., Papadopoulos, V., and Stavroulakis, G. AI-enhanced iterative solvers for accelerating the solution of large scale parametrized systems. July 2022. URL http://arxiv.org/abs/2207.02543.

Özbay, A. G., Hamzehloo, A., Laizet, S., Tzirakis, P., Rizos, G., and Schuller, B. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering*, 2, 2021. ISSN 2632-6736. doi: 10.1017/dce.2021.7.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Psaros, A. F., Kawaguchi, K., and Karniadakis, G. E. Meta-learning PINN loss functions. *Journal of computational physics*, 458:111121, June 2022. ISSN 0021-9991. doi: 10.1016/j.jcp.2022.111121. URL https://www.sciencedirect.com/science/article/pii/S0021999122001838.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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, February

2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Robertson, H. H. The solution of a set of reaction rate equations. In *Numerical Analysis: An introduction*, pp. 178–182. Academic Press, Cambridge, 1966.

Saad, Y. *Iterative Methods for Sparse Linear Systems: Second Edition*. Other Titles in Applied Mathematics. SIAM, April 2003. ISBN 9780898715347. doi: 10.1137/1.9780898718003. URL https://play.google.com/store/books/details?id=qtzmkzzqFmcC.

Shan, T., Tang, W., Dang, X., Li, M., Yang, F., Xu, S., and Wu, J. Study on a Fast Solver for Poisson's Equation Based on Deep Learning Technique. *IEEE transactions on antennas and propagation*, 68(9):6725–6733, September 2020. ISSN 0018-926X, 1558-2221. doi: 10.1109/TAP.2020.2985172. URL http://dx.doi.org/10.1109/TAP.2020.2985172.

Stanziola, A., Arridge, S. R., Cox, B. T., and Treeby, B. E. A Helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *Journal of computational physics*, 441:110430, September 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110430. URL https://www.sciencedirect.com/science/article/pii/S0021999121003259.

Taghibakhshi, A., MacLachlan, S., Olson, L., and West, M. Optimization-Based Algebraic Multigrid Coarsening Using Reinforcement Learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P. S., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 12129–12140. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/6531b32f8d02fece98ff36a64a7c8260-Paper.pdf.

Tang, W., Shan, T., Dang, X., Li, M., Yang, F., Xu, S., and Wu, J. Study on a Poisson's equation solver based on deep learning technique. In *2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, pp. 1–3, December 2017. doi: 10.1109/EDAPS.2017.8277017. URL http://dx.doi.org/10.1109/EDAPS.2017.8277017.

Um, K., Brand, R., Fei, Y. r., Holl, P., and Thuerey, N. Solver-in-the-loop: learning from differentiable physics to interact with iterative PDE-solvers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, number Article 513 in NIPS'20, pp. 6111–6122, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 9781713829546. URL https://dl.acm.org/doi/abs/10.5555/3495724.3496237.

Vaupel, Y., Hamacher, N. C., Caspari, A., Mhamdi, A., Kevrekidis, I. G., and Mitsos, A. Accelerating nonlinear model predictive control through machine learning. *Journal of process control*, 92:261–270, August 2020. ISSN 0959-1524. doi: 10.1016/j.jprocont.2020.06.012. URL https://www.sciencedirect.com/science/article/pii/S0959152420302481.

Venkataraman, S. and Amos, B. Neural Fixed-Point Acceleration for Convex Optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=Vxbpb6XvGgH.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, 17(3): 261–272, March 2020. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-019-0686-2. URL http://dx.doi.org/10.1038/s41592-019-0686-2.

Yang, S. and Gobbert, M. K. The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Applied mathematics letters*, 22(3):325–331, March 2009. ISSN 0893-9659. doi: 10.1016/j.aml.2008.03.028. URL https://www.sciencedirect.com/science/article/pii/S0893965908001523.

Yin, P., Lyu, J., Zhang, S., Osher, S. J., Qi, Y., and Xin, J. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Skh4jRcKQ.

Yonetani, R., Taniai, T., Barekatain, M., Nishimura, M., and Kanezaki, A. Path Planning using Neural A* Search. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12029–12039. PMLR, 2021. URL https://proceedings.mlr.press/v139/yonetani21a.html.

A. Organizing related works under meta-solving framework

Example 2 ((Cheng et al., 2021)). The target equation in (Cheng et al., 2021) is a linear system of equations $A_\eta u = f$ obtained by discretizing parameterized steady-state PDEs, where $u, f \in \mathbb{R}^N$ and $A_\eta \in \mathbb{R}^{N \times N}$ is determined by η, a parameter of the original equation. This work can be described in our framework as follows:

- Task τ: The task τ is to solve a linear system $A_\eta u = f$ for $\eta_= \eta_\tau$:
  - Dataset D_τ: The dataset D_τ is $\{\eta_\tau, f_\tau\}$.
  - Solution space $\mathcal{U}$: The solution space $\mathcal{U}$ is $\mathbb{R}^N$.
- Task space $(\mathcal{T}, P)$: The task distribution $(\mathcal{T}, P)$ is determined by the distribution of η_τ and f_τ.
- Loss function $\mathcal{L}$: The loss function $\mathcal{L} : \mathcal{T} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ is an unsupervised loss based on the residual of the equation, $\mathcal{L}(\tau, \hat{u}) = \|f_\tau - A_{\eta_\tau}\hat{u}\|^2 / \|f_\tau\|^2$.
- Solver Φ: The solver $\Phi : \mathcal{T} \times \Theta \to \mathcal{U}$ is iterations of a function $\phi_\tau(\cdot; \theta) : \mathcal{U} \to \mathcal{U}$ that represents an update step of the multigrid method. ϕ_τ is implemented using a convolutional neural network and its parameter θ is the weights corresponding to the smoother of the multigrid method. Note that weights of ϕ_τ other than θ are naturally determined by η_τ and the discretization scheme. In addition, ϕ_τ takes f_τ as part of its input at every step, but we write these dependencies as ϕ_τ for simplicity. To summarize, $\Phi(\tau; \theta) = \phi_\tau^k(u^{(0)}; \theta) = \hat{u}$, where k is the number of iterations of the multigrid method and $u^{(0)}$ is initial guess, which is $\mathbf{0}$ in the paper.
- Meta-solver Ψ: The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is implemented by a neural network with weights ω, which takes A_{η_τ} as its input and returns weights θ_τ that is used for the smoother inspired by the subspace correction method.

Example 3 ((Um et al., 2020)). In (Um et al., 2020), initial guesses for the Conjugate Gradient (CG) solver are generated by using a neural network. This work can be described in our framework as follows:

- Task τ: The task τ is to solve a pressure Poisson equation $\nabla \cdot \nabla p = \nabla \cdot v$ on 2D, where $p \in \mathbb{R}^{d_x \times d_y}$ is a pressure field and $v \in \mathbb{R}^{2 \times d_x \times d_y}$ is a velocity filed:
  - Dataset D_τ: The dataset D_τ is $D_\tau = \{v_\tau\}$, where v_τ is a given velocity sample.
  - Solution space $\mathcal{U}$: The solution space $\mathcal{U}$ is $\mathbb{R}^{d_x \times d_y}$.
- Task space $(\mathcal{T}, P)$: The task distribution $(\mathcal{T}, P)$ is determined by the distribution of v_τ.
- Loss function $\mathcal{L}$: The loss function $\mathcal{L}$ is $\mathcal{L}(\tau, \hat{p}) = \left\|\hat{p} - p^{(0)}\right\|^2$, where $\hat{p}$ is the approximate solution of the Poisson equation by the CG solver, and $p^{(0)}$ is the initial guess.
- Solver Φ: The solver $\Phi_k : \mathcal{T} \times \Theta \to \mathcal{U}$ is the differentiable CG solver with k iterations. Its parameter $\theta \in \Theta$ is the initial guess $p^{(0)}$, so $\Phi_k(\tau; \theta) = \hat{p}$.
- Meta-solver Ψ: The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is 2D U-Net with weights $\omega \in \Omega$, which takes v_τ as its input and returns the initigal guess $p_\tau^{(0)}$ for the CG solver.

Although the focus of this paper is on iterative algorithms, the meta-solving framework can describe other types of algorithms as well.

Example 4 ((Feliu-Fabà et al., 2020)). The authors in (Feliu-Fabà et al., 2020) propose the neural network architecture with meta-learning approach that solves the equations in the form $\mathcal{L}_\eta u(x) = f(x)$ with appropriate boundary conditions, where $\mathcal{L}_\eta$ is a partial differential or integral operator parametrized by a parameter function $\eta(x)$. This work can be described in our framework as follows:

- Task τ: The task τ is to solve a $\mathcal{L}_\eta u(x) = f(x)$ for $\eta = \eta_\tau$:
  - Dataset D_τ: The dataset D_τ is $D_\tau = \{\eta_\tau, f_\tau, u_\tau\}$, where $\eta_\tau, f_\tau, u_\tau \in \mathbb{R}^N$ are the parameter function, right hand side, and solution respectively.

  - Solution space $\mathcal{U}$: The solution space $\mathcal{U}$ is a subset of $\mathbb{R}^N$ for $N \in \mathbb{N}$.

- Task space $(\mathcal{T}, P)$: The task distribution $(\mathcal{T}, P)$ is determined by the distribution of η_τ and f_τ.

- Loss function $\mathcal{L}$: The loss function $\mathcal{L} : \mathcal{T} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ is the mean squared error with the reference solution, i.e. $\mathcal{L}(\tau, \hat{u}) = \|u_\tau - \hat{u}\|^2$.

- Solver Φ: The solver $\Phi : \mathcal{T} \times \Theta \to \mathcal{U}$ is implemented by a neural network imitating the wavelet transform, which is composed by three modules with weights $\theta = (\theta_1, \theta_2, \theta_3)$. In detail, the three modules, $\phi_1(\cdot; \theta_1)$, $\phi_2(\cdot; \theta_2)$, and $\phi_3(\cdot; \theta_3)$, represent forward wavelet transform, mapping η to the coefficients matrix of the wavelet transform, and inverse wavelet transform respectively. Then, using the modules, the solver Φ is represented by $\Phi(\tau; \theta) = \phi_3((\phi_2(\eta_\tau; \theta_2)\phi_1(f_\tau; \theta_1)); \theta_3) = \hat{u}$.

- Meta-solver Ψ: The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is the constant function that returns its parameter ω, so $\Psi(\tau; \omega) = \omega = \theta$ and $\Omega = \Theta$. Note that θ does not depend on τ in this example.

Example 5 ((Psaros et al., 2022)). In (Psaros et al., 2022), meta-learning is used for learning a loss function of the physics-informed neural network, shortly PINN (Raissi et al., 2019). The target equations are the following:

$$\mathcal{F}_\lambda[u](t, x) = 0, (t, x) \in [0, T] \times \mathcal{D} \tag{a}$$
$$\mathcal{B}_\lambda[u](t, x) = 0, (t, x) \in [0, T] \times \partial\mathcal{D} \tag{b}$$
$$u(0, x) = u_{0,\lambda}(x), x \in \mathcal{D}, \tag{c}$$

where $\mathcal{D} \subset \mathbb{R}^M$ is a bounded domain, $u : [0, T] \times \mathcal{D} \to \mathbb{R}^N$ is the solution, $\mathcal{F}_\lambda$ is a nonlinear operator containing differential operators, $\mathcal{B}_\lambda$ is a operator representing the boundary condition, $u_{0,\lambda} : \mathcal{D} \to \mathbb{R}^N$ represents the initial condition, and λ is a parameter of the equations.

- Task τ: The task τ is to solve a differential equation by PINN:
  - Dataset D_τ: The dataset D_τ is the set of points $(t, x) \in [0, T] \times \mathcal{D}$ and the values of u at the points if applicable. In detail, $D_\tau = D_{f,\tau} \cup D_{b,\tau} \cup D_{u_0,\tau} \cup D_{u,\tau}$, where $D_{f,\tau}, D_{b,\tau}$, and $D_{u_0,\tau}$ are sets of points corresponding to the equation Equation (a), Equation (b), and Equation (c) respectively. $D_{u,\tau}$ is the set of points (t, x) and observed values $u(t, x)$ at the points. In addition, each dataset $D_{\cdot,\tau}$ is divided into training set $D^{\text{train}}_{\cdot,\tau}$ and validation set $D^{\text{val}}_{\cdot,\tau}$.
  - Solution space $\mathcal{U}$: The solution space $\mathcal{U}$ is the weights space of PINN.

- Task space $(\mathcal{T}, P)$: The task distribution $(\mathcal{T}, P)$ is determined by the distribution of λ.

- Loss function $\mathcal{L}$: The loss function $\mathcal{L} : \mathcal{T} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ is based on the evaluations at the points in D^{val}_τ. In detail,

$$\mathcal{L}(\tau, \hat{u}) = L^{\text{val}}_\tau(\hat{u}) = L^{\text{val}}_{f,\tau}(\hat{u}) + L^{\text{val}}_{b,\tau}(\hat{u}) + L^{\text{val}}_{u_0,\tau}(\hat{u}),$$

  where

$$L^{\text{val}}_{f,\tau} = \frac{w_f}{|D_{f,\tau}|} \sum_{(t,x) \in D_{f,\tau}} \ell\left(\mathcal{F}_\lambda[\hat{u}](t, x), \mathbf{0}\right)$$
$$L^{\text{val}}_{b,\tau} = \frac{w_b}{|D_{b,\tau}|} \sum_{(t,x) \in D_{b,\tau}} \ell\left(\mathcal{B}_\lambda[\hat{u}](t, x), \mathbf{0}\right)$$
$$L^{\text{val}}_{u_0,\tau} = \frac{w_{u_0}}{|D_{u_0,\tau}|} \sum_{(t,x) \in D_{u_0,\tau}} \ell\left(\hat{u}(0, x), u_{0,\lambda}(x)\right),$$

  and $\ell : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_{\geq 0}$ is a function. In the paper, the mean squared error is used as ℓ.

- Solver Φ: The solver $\Phi : \mathcal{T} \times \Theta \to \mathcal{U}$ is the gradient descent for training the PINN. The parameter $\theta \in \Theta$ controls the objective of the gradient descent, $L^{\text{train}}_\tau(\hat{u}; \theta) = L^{\text{train}}_{f,\tau}(\hat{u}; \theta) + L^{\text{train}}_{b,\tau}(\hat{u}; \theta) + L^{\text{train}}_{u_0,\tau}(\hat{u}; \theta) + L^{\text{train}}_{u,\tau}(\hat{u}; \theta)$, where the difference from L^{val}_τ is that parametrized loss ℓ_θ is used in L^{train}_τ instead of the MSE in L^{val}_τ. Note that the loss weights w_f, w_b, w_{u_0}, w_u in L^{train}_τ are also considered as part of the parameter θ. In the paper, two designs of ℓ_θ are studied. One is using a neural network, and the other is using a learned adaptive loss function. In the former design, θ is the weights of the neural network, and in the latter design, θ is the parameter in the adaptive loss function.

- Meta-solver Ψ: The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is the constant function that returns its parameter ω, so $\Psi(\tau;\omega) = \omega = \theta$ and $\Omega = \Theta$. Note that θ does not depend on τ in this example.

Example 6 ((Yonetani et al., 2021)). In (Yonetani et al., 2021), the authors propose a learning-based search method, called Neural A*, for path plannning. This work can be described in our framework as follows:

- Task τ: The task τ is to solve a point-to-point shortest path problem on a graph $G = (V, E)$.
  - Dataset D_τ: The dataset D_τ is $\{G_\tau, v_{\tau,s}, v_{\tau,g}, p_\tau\}$, where $G_\tau = (V_\tau, E_\tau)$ is a graph (i.e. the environmental map of the task), $v_{\tau,s} \in V_\tau$ is a starting point, $v_{\tau,g} \in V_\tau$ is a goal, and p_τ is the ground truth path from $v_{\tau,s}$ to $v_{\tau,g}$.
  - Solution space $\mathcal{U}$: The solution space $\mathcal{U}$ is $\{0,1\}^{V_\tau}$.

- Task space $(\mathcal{T}, P)$: The task distribution $(\mathcal{T}, P)$ is determined according to each problem. The paper studies both synthetic and real-world datasets.

- Loss function $\mathcal{L}$: The loss function $\mathcal{L}$ is $\mathcal{L}(\tau, \hat{p}) = \|\hat{p} - p_\tau\|_1 / |V_\tau|$, where $\hat{p}$ is the search history by the A* algorithm. Note that the loss function considers not only the final solution but also the search history to improve the efficiency of the node explorations.

- Solver Φ: The solver $\Phi : \mathcal{T} \times \Theta \to \mathcal{U}$ is the differentiable A* algorithm proposed by the authors, which takes $D_\tau \setminus \{p_\tau\}$ and the guidance map $\theta_\tau \in \Theta = [0,1]^{V_\tau}$ that imposes a cost to each node $v \in V_\tau$, and returns the search history $\hat{p}$ containing the solution path.

- Meta-solver Ψ: The meta-solver $\Psi : \mathcal{T} \times \Omega \to \Theta$ is 2D U-Net with weights $\omega \in \Omega$, which takes $D_\tau \setminus \{p_\tau\}$ as its input and returns the guidance map θ_τ for the A* algorithm Φ.

B. Proofs in Section 2

We first present a lemma about the eigenvalues and eigenvectors of A and the corresponding Jacobi iteration matrix $M = I - \frac{1}{2}A$.

Lemma B.1. *The eigenvalues of A and M are $\mu_i = 2 - 2\cos\frac{i}{N+1}\pi$ and $\lambda_i = \cos\frac{i}{N+1}\pi$ for $i = 1, 2, \ldots, N$ respectively. Their common corresponding eigenvectors are $v_i = (\sin\frac{1}{N+1}i\pi, \sin\frac{2}{N+1}i\pi, \ldots, \sin\frac{N}{N+1}i\pi)^T$.*

Proof of Lemma B.1. The proof is presented in Section 9.1.1 of (Greenbaum, 1997). □

We can write down the loss functions, $\mathcal{L}_m$ and $\mathcal{L}_\epsilon$, and find their minimizers.

Proposition B.2 (Minimizer of (1))**.** *(1) is written as*

$$\min_{\omega}\ pc_1^2 (\omega\mu_1 - 1)^2 \lambda_1^{2m} + (1-p)c_2^2 (\omega\mu_2 - 1)^2 \lambda_2^{2m}, \tag{18}$$

and it has the unique minimizer

$$\omega_m = \frac{pc_1^2\mu_1\lambda_1^{2m} + (1-p)c_2^2\mu_2\lambda_2^{2m}}{pc_1^2\mu_1^2\lambda_1^{2m} + (1-p)c_2^2\mu_2^2\lambda_2^{2m}}. \tag{19}$$

Furthermore, for any $p \in (0,1)$, we have

$$\lim_{m\to\infty} \omega_m = \frac{1}{\mu_1}. \tag{20}$$

Proof of Proposition B.2. We have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tau_i \sim P}[\mathcal{L}_m(\tau_i, ; \omega)] &= p\mathcal{L}(\tau_1; \omega) + (1-p)\mathcal{L}(\tau_2; \omega) && (21)\\
&= p \left\| \hat{u}_1^{(m)}(\omega) - u_1^* \right\|^2 + (1-p) \left\| \hat{u}_2^{(m)}(\omega) - u_2^* \right\|^2 && (22)\\
&= p \left\| M^m(\hat{u}_1^{(0)}(\omega) - u_1^*) \right\|^2 + (1-p) \left\| M^m(\hat{u}_2^{(0)}(\omega) - u_2^*) \right\|^2 && (23)\\
&= p \left\| M^m(\omega c_1 \mu_1 v_1 - c_1 v_1) \right\|^2 + (1-p) \left\| M^m(\omega c_2 \mu_2 v_2 - c_2 v_2) \right\|^2 && (24)\\
&= p\lambda_1^{2m} c_1^2 (\omega\mu_1 - 1)^2 + (1-p)\lambda_2^{2m} c_2^2 (\omega\mu_2 - 1)^2. && (25)\\
& && (26)
\end{aligned}$$

Since $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_m(\tau_i, ; \omega)]$ is a quadratic function of ω, its minimum is achieved at

$$\omega_m = \frac{p c_1^2 \mu_1 \lambda_1^{2m} + (1-p) c_2^2 \mu_2 \lambda_2^{2m}}{p c_1^2 \mu_1^2 \lambda_1^{2m} + (1-p) c_2^2 \mu_2^2 \lambda_2^{2m}}. \tag{27}$$

Since $\lambda_1 > \lambda_2$, its limit is

$$\lim_{m \to \infty} \omega_m = \frac{p c_1^2 \mu_1}{p c_1^2 \mu_1^2} = \frac{1}{\mu_1}. \tag{28}$$

□

Proposition B.3 (Minimizer of (2)). *(2) is written as*

$$\min_{\omega} \; p \left(\frac{\log \frac{\epsilon}{c_1 |\omega\mu_1 - 1|}}{\log \lambda_1} \right)_+ + (1-p) \left(\frac{\log \frac{\epsilon}{c_2 |\omega\mu_2 - 1|}}{\log \lambda_2} \right)_+, \tag{29}$$

where $(x)_+ = \max\{0, x\}$. Assume $\epsilon < \frac{c_1 c_2 (\mu_2 - \mu_1)}{c_1 \mu_1 + c_2 \mu_2}$. If $p \neq p_\epsilon$, (29) has the unique minimizer

$$\omega_\epsilon = \begin{cases} \omega_{\epsilon,1} & \text{if } p > p_\epsilon \\ \omega_{\epsilon,2} & \text{if } p < p_\epsilon \end{cases} \tag{30}$$

where

$$\omega_{\epsilon,1} = \frac{1}{\mu_1} - \frac{\epsilon}{c_1 \mu_1}, \quad \omega_{\epsilon,2} = \frac{1}{\mu_2} + \frac{\epsilon}{c_2 \mu_2} \tag{31}$$

and

$$p_\epsilon = \frac{\log \lambda_1 \log \frac{\epsilon}{c_1(\omega_{\epsilon,1}\mu_2 - 1)}}{\log \lambda_1 \log \frac{\epsilon}{c_1(\omega_{\epsilon,1}\mu_2 - 1)} + \log \lambda_2 \log \frac{\epsilon}{c_2(1 - \omega_{\epsilon,2}\mu_1)}}. \tag{32}$$

If $p = p_\epsilon$, (29) has two different minimizers $\omega_{\epsilon,1}$ and $\omega_{\epsilon,2}$. Furthermore, we have

$$\lim_{\epsilon \to 0} \omega_\epsilon = \begin{cases} \frac{1}{\mu_1} & \text{if } p > p_0 \\ \frac{1}{\mu_2} & \text{if } p < p_0 \end{cases}, \tag{33}$$

where

$$p_0 = \lim_{\epsilon \to 0} p_\epsilon = \frac{\log \lambda_1}{\log \lambda_1 + \log \lambda_2}. \tag{34}$$

Proof of Proposition B.3. We consider the relaxed version $\mathcal{L}_\epsilon(\tau; \omega) = \min\{m \in \mathbb{R}_{\geq 0} : \mathcal{L}_m(\tau; \omega) \leq \epsilon^2\}$. By solving $\mathcal{L}_n(\tau_i; \omega) = \epsilon^2$ for n, we have

$$\mathcal{L}_\epsilon(\tau_i; \omega) = \max \left\{ 0, \frac{\log \frac{\epsilon}{c_i |\omega\mu_i - 1|}}{\log \lambda_i} \right\}, \tag{35}$$

which deduces

$$\mathop{\mathbb{E}}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)] = p \max\left\{0, \frac{\log \frac{\epsilon}{c_1|\omega\mu_1 - 1|}}{\log \lambda_1}\right\} + (1-p)\max\left\{0, \frac{\log \frac{\epsilon}{c_2|\omega\mu_2 - 1|}}{\log \lambda_2}\right\}. \tag{36}$$

Assuming $\epsilon < \frac{c_1 c_2(\mu_2 - \mu_1)}{c_1\mu_1 + c_2\mu_2}$, this can be written as

$$\mathop{\mathbb{E}}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)] = \begin{cases} p\frac{\log \frac{\epsilon}{c_1(1-\omega\mu_1)}}{\log \lambda_1} & \text{if } \omega \in \left[\frac{1}{\mu_2} - \frac{\epsilon}{c_2\mu_2}, \frac{1}{\mu_2} + \frac{\epsilon}{c_2\mu_2}\right] \\ (1-p)\frac{\log \frac{\epsilon}{c_2(\omega\mu_2 - 1)}}{\log \lambda_2} & \text{if } \omega \in \left[\frac{1}{\mu_1} - \frac{\epsilon}{c_1\mu_1}, \frac{1}{\mu_1} + \frac{\epsilon}{c_1\mu_1}\right] \\ p\frac{\log \frac{\epsilon}{c_1|\omega\mu_1 - 1|}}{\log \lambda_1} + (1-p)\frac{\log \frac{\epsilon}{c_2|\omega\mu_2 - 1|}}{\log \lambda_2} & \text{otherwise.} \end{cases} \tag{37}$$

Note that $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$ is strictly decreasing for $\omega \in (-\infty, \frac{1}{\mu_2} + \frac{\epsilon}{c_2\mu_2}]$ and strictly increasing for $\omega \in [\frac{1}{\mu_1} - \frac{\epsilon}{c_1\mu_1}, \infty)$. Since $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$ is concave for $\omega \in [\frac{1}{\mu_2} + \frac{\epsilon}{c_2\mu_2}, \frac{1}{\mu_1} - \frac{\epsilon}{c_1\mu_1}]$, its minimum is attained at $\omega_{\epsilon,1} = \frac{1}{\mu_1} - \frac{\epsilon}{c\mu_1}$ or $\omega_{\epsilon,2} = \frac{1}{\mu_2} + \frac{\epsilon}{c_2\mu_2}$. Then, by comparing $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_{\epsilon,2})] = p\mathcal{L}_\epsilon(\tau_1;\omega_{\epsilon,2})$ and $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_{\epsilon,1})] = (1-p)\mathcal{L}_\epsilon(\tau_2;\omega_{\epsilon,1})$, we can deduce the result. □

Let us prove Proposition 3.1 first before Proposition 2.2.

Proof of Proposition 3.1. The limits are shown in Proposition B.3.

We now show the second part. Note that if $\delta_1 > \delta_2 \geq \epsilon > 0$, then $p_0 < p_\epsilon \leq p_{\delta_2} < p_{\delta_1}$. Let $\omega_{\max} = \text{argmax}_{\omega \in [1/\omega_{\epsilon,2}, 1/\omega_{\epsilon,1}]} \mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$.

If $p < p_0$, then we have $\omega_\epsilon = \omega_{\epsilon,2} \leq \omega_{\delta_2} = \omega_{\delta_2,2} < \omega_{\delta_1} = \omega_{\delta_1,2}$. Note that $p < p_0$ and $\delta_1 < \frac{c_1c_2(\mu_2-\mu_1)}{2(c_1\mu_1 + c_2\mu_2)}$ guarantee $\omega_{\delta_1} < \omega_{\max}$. Since $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$ is increasing for $\omega \in [\omega_\epsilon, \omega_{\max}]$ and $\omega_\epsilon \leq \omega_{\delta_2} < \omega_{\delta_1} < \omega_{\max}$, we have $\mathbb{E}_{\tau \sim P}[L_\epsilon(\tau;\omega_{\delta_1})] > \mathbb{E}_{\tau \sim P}[L_\epsilon(\tau;\omega_{\delta_2})]$.

If $p > p_{\delta_1}$, then we have $\omega_\epsilon = \omega_{\epsilon,1} \geq \omega_{\delta_2} = \omega_{\delta_2,1} > \omega_{\delta_1} = \omega_{\delta_1,1}$. Note that $p > p_0$ and $\delta_1 < \frac{c_1c_2(\mu_2-\mu_1)}{2(c_1\mu_1 + c_2\mu_2)}$ guarantee $\omega_{\delta_1} > \omega_{\max}$. Since $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$ is decreasing for $\omega \in [\omega_{\max}, \omega_\epsilon]$ and $\omega_\epsilon \geq \omega_{\delta_2} > \omega_{\delta_1} > \omega_{\max}$, we have $\mathbb{E}_{\tau \sim P}[L_\epsilon(\tau;\omega_{\delta_1})] > \mathbb{E}_{\tau \sim P}[L_\epsilon(\tau;\omega_{\delta_2})]$.

We now show the last part. Recall that $\omega_m = \frac{pc_1^2\mu_1\lambda_1^{2m} + (1-p)c_2^2\mu_2\lambda_2^{2m}}{pc_1^2\mu_1^2\lambda_1^{2m} + (1-p)c_2^2\mu_2^2\lambda_2^{2m}}$. Setting $m = 0$, we have $\omega_0 = \frac{pc_1^2\mu_1 + (1-p)c_2^2\mu_2}{pc_1^2\mu_1^2 + (1-p)c_2^2\mu_2^2}$. For $\delta > 0$, we take c_1, c_2, and p so that $\omega_\epsilon \leq \omega_\delta < \omega_{\max} < \omega_0 < \omega_m$ and $\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_\delta)] > 0$. For example, $c_1 = \frac{\mu_2(\log\lambda_1 + \log\lambda_2)}{p^2(\mu_2-\mu_1)\log\lambda_2}\delta$, $c_2 = \frac{\mu_1(\log\lambda_1 + \log\lambda_2)}{p(\mu_2-\mu_1)\log\lambda_1}\delta$, and $p < p_0$ satisfy the relationship. Then, we have

$$\begin{aligned} \frac{\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_m)]}{\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_\delta)]} &= \frac{p\mathcal{L}_\epsilon(\tau_1;\omega_m) + (1-p)\mathcal{L}_\epsilon(\tau_2;\omega_m)}{p\mathcal{L}_\epsilon(\tau_1;\omega_\delta) + (1-p)\mathcal{L}_\epsilon(\tau_2;\omega_\delta)} & (38) \\ &\geq \frac{(1-p)\mathcal{L}_\epsilon(\tau_2;\omega_m)}{p\mathcal{L}_\epsilon(\tau_1;\omega_\delta) + (1-p)\mathcal{L}_\epsilon(\tau_2;\omega_\delta)} & (39) \\ &\geq \frac{(1-p)\mathcal{L}_\epsilon(\tau_2;\omega_0)}{p\mathcal{L}_\epsilon(\tau_1;\omega_\delta) + (1-p)\mathcal{L}_\epsilon(\tau_2;\omega_\delta)} & (40) \\ &= \frac{(1-p)\log\lambda_1 \log\frac{\epsilon}{c_2(\omega_0\mu_2 - 1)}}{p\log\lambda_2\log\frac{\epsilon}{c_1(1-\omega_\delta\mu_1)} + (1-p)\log\lambda_1\log\frac{\epsilon}{c_2(\omega_\delta\mu_2 - 1)}} & (41) \\ & & (42) \end{aligned}$$

Substituting ω_0, ω_δ, c_1, and c_2 and taking the limit as $p \to 0$, we have $\frac{\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_m)]}{\mathbb{E}_{\tau_i \sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_\delta)]} \to \infty$.

□

Proof of Proposition 2.2. The limits are shown in Proposition B.2.

We now show the second part. Since ω_m is increasing in m and $\lim_{m\to\infty}\omega_m = 1/\mu_1$, there exists m_0 such that $\omega_{\epsilon,1} < \omega_{m_0}$. For any m_1 and m_2, if $m_0 < m_1 < m_2$, then $\omega_{m_0} < \omega_{m_1} < \omega_{m_2}$. Hence, $\mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_{m_1})] < \mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_{m_2})]$ because $\mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega)]$ is increasing for $\omega \in [\omega_{\epsilon,1}, 1/\mu_1]$.

For the last part, the proof is similar to Proposition 3.1. Take $c_1 = \frac{\mu_2(\log\lambda_1+\log\lambda_2)}{p^2(\mu_2-\mu_1)\log\lambda_2}\epsilon$, $c_2 = \frac{\mu_1(\log\lambda_1+\log\lambda_2)}{p(\mu_2-\mu_1)\log\lambda_1}\epsilon$, and $p < p_0$ and substitute them into $\mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_m)]$. Then, we have $\mathbb{E}_{\tau_i\sim P}[\mathcal{L}_\epsilon(\tau_i;\omega_m)] \to \infty$ as $p \to 0$.

$\square$

C. Details of numerical examples

C.1. Details in Section 2.2.1 and Section 3.1.2

Task In distribution P_1, f_τ is represented by

$$f_\tau = \sum_{i=1}^{N} c_i\mu_i v_i, \text{ where } c_i \sim \mathcal{N}(0, \left|\frac{N+1-2i}{N-1}\right|). \tag{43}$$

In distribution P_2, f_τ is represented by

$$f_\tau = \sum_{i=1}^{N} c_i\mu_i v_i, \text{ where } c_i \sim \mathcal{N}(0, 1-\left|\frac{N+1-2i}{N-1}\right|). \tag{44}$$

The discretization size is $N = 16$ in the experiments. The number of tasks for training, validation, and test are all 10^3.

Network architecture and hyper-parameters In Section 2.2.1 and Section 3.1.2, Ψ_{nn} is a fully connected neural network with two hidden layers of 15 neurons. Its input is discretized $f_\tau \in \mathbb{R}^N$ and the output is c_i's of $\hat{u}^{(0)} = \sum_{i=1}^{N} c_i v_i$. The activation function is SiLU (Elfwing et al., 2018) for hidden layers. The optimizer is Adam (Kingma & Ba, 2015) with learning rate 0.01 and $(\beta_1,\beta_2) = (0.999, 0.999)$. The batch size is 256. The model is trained for 2500 epochs. During training, if the validation loss does not decrease for 100 epochs, the learning rate is reduced by a factor of $1/5$. The presented results are obtained by the best models selected based on the validation loss.

Results The detailed results of Figure 1 are presented in Table 3.

Table 3. The average number of iterations for solving Poisson equations. Boldface indicates the best performance for each column.

			$p=0$				$p=0.01$			
			ϵ (test)							
Ψ	m (train)	ϵ (train)	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-2}	10^{-4}	10^{-6}	10^{-8}
Ψ_{base}	-	-	28.17	92.54	158.41	224.28	30.15	96.52	164.41	232.30
Ψ_{nn}	0	-	**0.00**	17.52	168.80	436.95	2.11	183.28	451.24	719.40
	1	-	**0.00**	20.07	148.26	403.22	1.90	178.61	446.51	714.67
	5	-	1.00	13.82	135.43	397.97	1.18	34.47	286.43	554.55
	25	-	6.26	16.31	37.71	172.02	7.11	20.22	121.24	379.51
	125	-	23.42	83.28	148.73	214.59	20.96	80.44	151.39	353.35
	-	10^{-2}	0.01	40.31	223.15	491.30	**0.25**	185.93	453.84	721.99
	-	10^{-4}	3.93	**9.76**	48.61	178.15	5.34	**13.69**	176.77	444.71
	-	10^{-6}	6.20	15.79	**32.00**	**83.43**	7.94	19.79	**36.11**	166.30
	-	10^{-8}	11.12	32.05	57.62	86.70	13.39	37.44	66.16	**97.97**

C.2. Details in Section 4.1

Task We prepare $10,000$ sets of c_1, c_2, c_3 and use $2,500$ for training, $2,500$ for validation, and $5,000$ for test. Since the solution of the Robertson equation has a quick initial transient followed by a smooth variation (Hairer & Wanner, 2010), we set the step size h_n ($n = 1, 2, \dots, 100$) so that the evaluation points are located log-uniformly in $[10^{-6}, 10^3]$. Thus, each set of c_1, c_2, c_3 is associated with 100 data points.

Solver The iteration rule of the Newton-SOR method is

$$J_n(y_{n+1}^{(k)}) = D_k - L_k - U_k \tag{45}$$
$$y_{n+1}^{(k+1)} = y_{n+1}^{(k)} - r(D_k - rL_k)^{-1} g_n(y_{n+1}^{(k)}), \tag{46}$$

where J_n is the Jacobian of g_n, its decomposition D_k, L_k, U_k are diagonal, strictly lower triangular, and strictly upper triangular matrices respectively, and $r \in [1, 2]$ is the relaxation factor of the SOR method. It iterates until the residual of the approximate solution $\left\|g_n(y_{n+1}^{(k)})\right\|$ reaches a given error tolerance ϵ. Note that we choose to investigate the Newton-SOR method, because it is a fast and scalable method and applicable to larger problems.

Network architecture and hyper-parameters In Section 4.1, Ψ_{ini}, Ψ_{relax}, and Ψ_{both} are fully-connected neural networks that take c_1, c_2, c_3, h_n, y_n as an input. They have two hidden layers with 1024 neurons, and ReLU is used as the activation function except for the last layer. The difference among them is only the last layer. The last layer of Ψ_{ini} modifies the previous timestep solution y_n for a better initial guess $y_\tau \in \mathbb{R}^3$. Since the scale of each element of y_n is quite different, the modification is conducted in log scale, i.e.

$$y_\tau = \exp(\log(y_n + \tanh(W_{\text{ini}} x))), \tag{47}$$

where x is the input of the last layer and W_{ini} are its weights. The last layer of Ψ_{relax} is designed to output the relaxation factor $r_\tau \in [1, 2]$:

$$r_\tau = \text{sigmoid}(W_{\text{relax}} x) + 1, \tag{48}$$

where x is the input of the last layer and W_{relax} are its weights. The last layer of Ψ_{both} is their combination.

The meta-solvers are trained for 200 epochs by Adam with batchsize 16384. The initial learning rate is $2.0 \cdot 10^{-5}$, and it is reduced at the 100th epoch and 150th epoch by $1/5$. For Ψ_{relax} and Ψ_{both}, the bias in the last layer corresponding to the relaxation factor is set to -1 at the initialization for preventing unstable behavior in the beginning of the training.

C.3. Details in Section 4.2

Task To set up problem instances, open-source computing library FEniCS (Alnæs et al., 2015) is used. We sampled tasks $1,000$ for training, $1,000$ for validation, and $1,000$ for test.

Solver Our implementation of the ICCG method is based on (Huang et al., 2009).

Network architecture and hyper-parameters In Section 4.2, Ψ_{nn} is a fully-connected neural network with two hidden layers of 128 neurons. Its input is $\{W_\tau, \lambda_\tau, \mu_\tau, \rho_\tau, \|A_\tau\|_1, \|A_\tau\|_\infty, \|A_\tau\|_F\}$ and output is diagonal shift α_τ. The activation function is ReLU except for the last layer. At the last layer, the sigmoid function is used as the activation function. The hyper-parameter γ in the loss functions is $\gamma = 500$.

The meta-solver is trained for 200 epochs by Adam with batchsize 128. The initial learning rate is 0.001 and it is reduced at the 100th epoch and 150th epoch by $1/5$. The presented results are obtained by the best models selected based on the validation loss.