Meta Learning of Interface Conditions for Multi-Domain Physics-Informed Neural Networks

Shibo Li [*1] **Michael Penwarden** [*1 2] **Yiming Xu** [3] **Conor Tillinghast** [3] **Akil Narayan** [2 3] **Robert Kirby** [1 2] **Shandian Zhe** [1]

Abstract

Physics-informed neural networks (PINNs) are emerging as popular mesh-free solvers for partial differential equations (PDEs). Recent extensions decompose the domain, apply different PINNs to solve the problem in each subdomain, and stitch the subdomains at the interface. Thereby, they can further alleviate the problem complexity, reduce the computational cost, and allow parallelization. However, the performance of multi-domain PINNs is sensitive to the choice of the interface conditions. While quite a few conditions have been proposed, there is no suggestion about how to select the conditions according to specific problems. To address this gap, we propose META Learning of Interface Conditions (METALIC), a simple, efficient yet powerful approach to dynamically determine appropriate interface conditions for solving a family of parametric PDEs. Specifically, we develop two contextual multi-arm bandit (MAB) models. The first one applies to the entire training course, and online updates a Gaussian process (GP) reward that given the PDE parameters and interface conditions predicts the performance. We prove a sub-linear regret bound for both UCB and Thompson sampling, which in theory guarantees the effectiveness of our MAB. The second one partitions the training into two stages, one is the stochastic phase and the other deterministic phase; we update a GP reward for each phase to enable different condition selections at the two stages to further bolster the flexibility and performance. We have shown the advantage of METALIC on four bench-mark PDE families.

[*]Equal contribution [1]Kahlert School of Computing, University of Utah [2]Scientific Computing and Imaging (SCI) Institute, University of Utah [3]Department of Mathematics, University of Utah. Correspondence to: Shandian Zhe <zhe@cs.utah.edu>.

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

1 Introduction

Physics-informed neural networks (PINNs) (Raissi et al., 2019) have become a popular mesh-free approach for solving partial differential equations (PDEs). They have shown successful in many scientific and engineering problems, *e.g.*, (Sahli Costabal et al., 2020; Kissas et al., 2020; Sun et al., 2020). The recent multi-domain versions, *e.g.*, (Jagtap et al., 2020; Jagtap and Karniadakis, 2021), extend PINNs with a divide-and-conquer strategy, and have attracted considerable attention. Specifically, they decompose the domain of interest into several subdomains, place a separate PINN to solve the PDE at each subdomain, and stitch the subdomains at the interface. In this way, the multi-domain PINNs can alleviate problem complexity, adopt simpler architectures, reduce the training cost, and enable parallel computation (Shukla et al., 2021).

However, the performance of multi-domain PINNs is sensitive to the choice of interface conditions (or regularizers) that unify the PINN solutions across different subdomains. Quite a few conditions have been proposed, such as those that encourage solution and residual continuity (Jagtap and Karniadakis, 2021), flux conservation (Jagtap et al., 2020), gradient continuity (De Ryck et al., 2022) and residual gradient continuity (Yu et al., 2022). On one hand, naively combining all possible interface conditions does not necessarily give the optimal performance; instead, it can complicate the loss landscape, making optimization more costly and challenging. On the other hand, the best conditions can vary across problems, which are up to the problem properties. Currently, there is no suggestion about how to select interface conditions according to specific problems, bringing inconvenience and difficulties to the multi-domain practice.

To address this issue, in this paper we propose METALIC, a simple, efficient and powerful method that can dynamically determine the appropriate interface conditions for solving a family of parametric PDEs. The contributions of our work are summarized as follows.

- **Problem Formulation and Strategy.** We formulate interface selection as a novel meta learning problem, and propose to use the multi-arm bandit (MAB) frame-

work to address the problem. Compared with other complex/popular approaches, *e.g.*, reinforcement learning with policy gradients, MAB is simple and efficient, requiring much less training trajectories and almost no hyper-parameter tuning. Due to the online nature, we can apply the learned MAB to select the conditions for solving new PDEs while continuously improving the model. To our knowledge, this is the first time of using MAB to address important meta learning tasks.

- **Method.** We develop two contextual MAB models, where we view the PDE parameters as the context, sets of interface conditions as the arms, and the solution accuracy as the reward. The first model applies to the entire training course, and online updates a Gaussian process (GP) reward model with Upper Confidence Bound (UCB) or Thompson sampling (TS). The second model, according to the common PINN practice, partitions the training into two phases, the stochastic (ADAM) and deterministic (LBFGS) phase. We sequentially learn a separate contextual bandit for each phase. In this way, we can select appropriate conditions for different training stages to enhance flexibility and to further improve the performance.
- **Theory.** We prove that our first MAB model, under the PDE parameters context and with our mixed continuous and categorical kernel, enjoys a sublinear regret bound with both UCB and TS. It means that over the course of online training, our MAB enables increasingly better interface condition selections, and guarantees to eventually find the optimal conditions.
- **Results and Analysis.** We evaluated METALIC with four commonly used benchmark PDEs in PINN literature. Both of our MAB models exhibit a sublinear growth of the accumulated solution error, which is much slower than the linear growth of random arm selection. We then examined METALIC after online training. With the interface conditions determined by METALIC, the multi-domain PINNs achieve solution errors one order of magnitude smaller than using randomly selected conditions, which also outperform standard single-domain PINNs with more neurons (we obtained as much as 88.5% error reduction). Finally, we conducted a thorough analysis of the conditions selected by METALIC and found many interesting results. Many selected conditions not only reflect the physical properties of the problem, but also are tied to the specific optimization procedure for PINN training.

2 Background

Physics-Informed Neural Networks (PINNs) estimate PDE solutions with (deep) neural networks. Consider a PDE of the following general form,

$$\begin{aligned} \mathcal{F}[u](\mathbf{x}) &= h(\mathbf{x}), \quad \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \end{aligned} \tag{1}$$

where $\mathcal{F}$ is the differential operator for the PDE, Ω is the domain, $\partial\Omega$ is the boundary of the domain, $h : \Omega \to \mathbb{R}$, and $g : \partial\Omega \to \mathbb{R}$ are the given source and boundary functions, respectively. To solve the PDE, the PINN uses a deep neural network $\widehat{u}_{\boldsymbol{\theta}}(\mathbf{x})$ to represent the solution u, samples N collocation points $\{\mathbf{x}_c^i\}_{i=1}^N$ from Ω and M points $\{\mathbf{x}_b^i\}_{i=1}^M$ from $\partial\Omega$, and minimizes the loss,

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \ \lambda_b L_b(\boldsymbol{\theta}) + L_r(\boldsymbol{\theta}), \tag{2}$$

where $L_b(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^M \left(\widehat{u}_{\boldsymbol{\theta}}(\mathbf{x}_b^i) - g(\mathbf{x}_b^i)\right)^2$ is the boundary term to fit the boundary condition, $L_r(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^N \left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}}](\mathbf{x}_c^i) - h(\mathbf{x}_c^i)\right)^2$ is the residual term to fit the equation, and $\lambda_b > 0$ is the weight of the boundary term. One can also add an initial condition term in the loss function to fit the initial conditions (when needed).

Multi-Domain PINNs decompose the domain Ω into several subdomains $\Omega_1, \ldots, \Omega_K$, and assign a separate PINN $\widehat{u}_{\boldsymbol{\theta}_k}$ to solve the PDE in each subdomain Ω_k. The loss for each PINN includes a boundary term $L_b^k(\boldsymbol{\theta}_k)$ and residual term $L_r^k(\boldsymbol{\theta}_k)$ similar to those in (2), based on the boundary and collocation points sampled from $\partial\Omega_k$ and Ω_k, respectively. In addition, to stitch together the subdomains to obtain an overall solution over Ω, we introduce interface conditions into the loss to align different PINNs at the intersection of the subdomains. There have been quite a few interface conditions. Suppose $\Omega_k \cap \Omega_{k'} \neq \emptyset$. We sample a set of interface points $\{\mathbf{x}_{k,k'}^i\}_{i=1}^{J_{k,k'}} \in \Omega_k \cap \Omega_{k'}$. One commonly used interface condition is to encourage the solution continuity (Jagtap and Karniadakis, 2021),

$$I_1(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}_{k,k'}^i) - \widehat{u}_{k,k'}^{\text{avg}}(\mathbf{x}_{k,k'}^i)\right)^2$$

where $\widehat{u}_{k,k'}^{\text{avg}}(\mathbf{x}_{k,k'}^i) = \frac{1}{2}\left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}_{k,k'}^i) + \widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}_{k,k'}^i)\right)$. Other choices include the residual continuity (Jagtap and Karniadakis, 2021), gradient continuity (De Ryck et al., 2022), residual gradient continuity (Yu et al., 2022), flux conservation (Jagtap et al., 2020), *etc*. In general, the loss for each subdomain k has the following form,

$$\mathcal{L}^k = \lambda_b L_b^k(\boldsymbol{\theta}_k) + L_r^k(\boldsymbol{\theta}_k) + \lambda_I \sum_{k':\Omega_{k'}\cap\Omega_k\neq\emptyset} \sum_{n\in\mathcal{S}} I_n(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'})$$

where $\mathcal{S}$ is the set of interface conditions, and $\lambda_I > 0$ is the weight of the interface term. The training is to minimize $\mathcal{L} = \sum_{k=1}^K \mathcal{L}^k$. The final solution inside each sub-domain k is given by the associated PINN $\widehat{u}_{\boldsymbol{\theta}_k}$, while on the interface, by the average of the PINNs that share the interface.

3 Meta Learning of Interface Conditions

While multi-domain PINNs have shown successes, the selection of the interface conditions remains an open and difficult problem. On one hand, naively adding all possible conditions together will complicate the loss landscape, making the optimization challenging and expensive, yet not necessarily giving the best performance. On the other hand, different problems can demand a different set of the interface conditions as the best choice, which is up to the properties of the problem itself. For example, in our preliminary study about a specific 2D Poisson equation (see Sec C in Appendix), we found the best accuracy is obtained with a novel combination of three interface conditions, giving 43.7% error reduction as compared with combining all the interface conditions together; see Table 5 in Appendix.

Currently, there is a lack of methodologies to identify conditions for different PDEs. To address this issue, we first formulate it as a novel meta learning problem. Specifically, we consider a parametric PDE family $\mathcal{A}$, where each PDE in $\mathcal{A}$ is parameterized by $\boldsymbol{\beta} \in \mathcal{X} \subset \mathbb{R}^d$. The parameters can come from the operator $\mathcal{F}$, the source term h and/or the boundary function g (see (1)). Denote by $\mathcal{S} = \{I_1, \ldots, I_s\}$ the full set of interface conditions. Our goal is, given a PDE parameterized by arbitrary $\boldsymbol{\beta} \in \mathcal{X}$, to determine $I(\boldsymbol{\beta}) \subseteq \mathcal{S}$ — the best set of interface conditions — for multi-domain PINNs to solve that PDE.

We propose to use the multi-arm bandit (MAB) framework (Slivkins et al., 2019) to address this problem. One might consider other complex and prevalent approaches, such as deep neural network prediction and reinforcement learning with policy gradients. However, to get well trained, these methods usually demand massive running trajectories of PINNs, which can be extremely costly. In addition, the success of these methods also rely on elaborate architecture design and intensive tuning of many hyper-parameters. By contrast, MAB is simple and efficient, requiring much less training trajectories and (almost) no architecture design and hyper-parameter tuning. The online nature makes the MAB straightforward to update incrementally with new data, and is much more convenient than those heavy-duty models.

3.1 Multi-Arm Bandit for Entire Training

We first propose a MAB model to select the interface conditions for the entire training procedure of multi-domain PINNs. In general, MAB considers a gambler playing q slot machines (*i.e.*, arms). Pulling the lever of each machine will return a random reward from a machine-specific probabilistic distribution, which is unknown apriori. The gambler aims to maximize the total reward earned from a series of lever-pulls across the q machines. For each play, the gambler needs to decide the tradeoff between exploiting the machine that has observed the largest expected payoff so far and exploring the payoffs of other machines. To determine PDE-specific interface conditions, we build a contextual MAB model. We consider the PDE parameters $\boldsymbol{\beta} \in \mathcal{X}$ as the context, all possible combinations of the interface conditions (*i.e.*, the power set of $\mathcal{S}$) as the arms, and the negative solution error as the reward. The problem space can be represented by a triplet $(\mathcal{X}, \mathcal{P}, f(\cdot,\cdot))$, where $\mathcal{X}$ is the context space, $\mathcal{P}$ is the action space (the power set of $\mathcal{S}$), and $f : \mathcal{X} \times \mathcal{P} \to \mathbb{R}$ is the reward function. We represent each action by an s-dimensional binary vector $\mathbf{a}$, where each element corresponds to a particular interface condition in $\mathcal{S}$. The i-th element $a_i = 1$ means the interface condition i is selected in the action.

To estimate the unknown reward function $f(\cdot,\cdot)$, we assign a Gaussian process (GP) prior,

$$f \sim \mathcal{GP}\left(0, \kappa\left([\boldsymbol{\beta}, \mathbf{a}], [\boldsymbol{\beta}', \mathbf{a}']\right)\right) \tag{3}$$

where $\kappa(\cdot,\cdot)$ is a kernel (covariance) function. Considering the categorical nature of the action input, we design a product kernel,

$$\kappa\left([\boldsymbol{\beta}, \mathbf{a}], [\boldsymbol{\beta}', \mathbf{a}']\right) = \kappa_1(\boldsymbol{\beta}, \boldsymbol{\beta}')\kappa_2(\mathbf{a}, \mathbf{a}') \tag{4}$$

where $\kappa_1(\boldsymbol{\beta}, \boldsymbol{\beta}') = \exp(-\tau_1 \|\boldsymbol{\beta} - \boldsymbol{\beta}'\|^2)$ is the square exponential (SE) kernel for continuous PDE parameters, and

$$\kappa_2(\mathbf{a}, \mathbf{a}') = \exp\left(\tau_2 \cdot \frac{1}{s} \sum_{i=1}^{s} \mathbb{1}(a_i = a_i')\right) \tag{5}$$

where $\mathbb{1}(\cdot)$ is the indicator function. Hence, the similarity between actions is based on the overlap ratio of the selected interface conditions, which is natural and intuitive. As in the standard GP regression, the observed reward r is then sampled from a Gaussian noise model,

$$r \sim \mathcal{N}(r|f, \sigma_0^2) \tag{6}$$

where $\sigma_0^2 > 0$. The noise model is to capture the extraneous randomness such as stochastic training and float rounding.

To learn the MAB, each step we randomly sample a context $\boldsymbol{\beta}$ from $\mathcal{X}$, and then select an action $\mathbf{a}$, *i.e.*, a set of interface conditions, according to the current GP reward model. We then run the multi-domain PINNs with the interface conditions to solve the PDE parameterized by $\boldsymbol{\beta}$. We evaluate the negative solution error ξ as the received reward. We add the new data point $([\boldsymbol{\beta}, \mathbf{a}], -\xi)$ into the current training set, and retrain (update) the GP reward model. We repeat this procedure until a given maximum number of trials (plays) is done. To fulfill a good exploration-exploitation tradeoff, we use the Upper Confidence Bound (UCB) (Auer, 2002; Srinivas et al., 2010) or Thompson sampling (TS) (Thompson, 1933; Chapelle and Li, 2011) to select the action at each

step. Specifically, denote the current predictive distribution of the GP surrogate by

$$p(\widehat{f}|\mathcal{D}, \boldsymbol{\beta}, \mathbf{a}) = \mathcal{N}\left(\widehat{f}|\mu(\mathbf{a}, \boldsymbol{\beta}), \sigma^2(\mathbf{a}, \boldsymbol{\beta})\right)$$

where $\mathcal{D}$ is the accumulated training set so far. The UCB score is

$$\mathrm{UCB}(\mathbf{a}) = \mu(\mathbf{a}, \boldsymbol{\beta}) + {c_t}^{1/2} \cdot \sigma(\mathbf{a}, \boldsymbol{\beta})$$

where $c_t > 0$ is a coefficient at step t, and the TS score is sampled from the predictive distribution,

$$\mathrm{TS}(\mathbf{a}) \sim p(\widehat{f}|\mathcal{D}, \boldsymbol{\beta}, \mathbf{a}).$$

We can see that both scores integrate the predictive mean (which reflects the exploitation part) and the variance information (exploration part). We evaluate the score for each action $\mathbf{a} \in \mathcal{P}$ (UCB or TS), and select the one with the highest score (corresponding to the best tradeoff).

In the online scenario, we can keep using UCB or TS to select the action for incoming new PDEs while improving the reward model according to the measured solution error. When the online playing is done and we no longer conduct exploration to update our model, given a new PDE (say, indexed by $\boldsymbol{\beta}^*$), we evaluate the predictive mean μ given $\boldsymbol{\beta}^*$ and every $\mathbf{a} \in \mathcal{P}$. We then select the one with the largest predictive mean (reward estimate). We use the corresponding interface conditions to run the multi-domain PINNs to solve the PDE. Our MAB learning is summarized in Algorithm 1.

To ensure our MAB is capable of finding the optimal interface conditions for different PDEs, we perform regret analysis. Consider a sequence of PDE parameters (*i.e.*, context) in the online playing, $\{\boldsymbol{\beta}_t\}$. Denote by $\mathbf{a}_t^*$ and $\mathbf{a}_t$ the optimal and MAB-selected interface condition set for each $\boldsymbol{\beta}_t$, respectively. We define an instantaneous regret $\zeta_t = f(\boldsymbol{\beta}_t, \mathbf{a}_t) - f(\boldsymbol{\beta}_t, \mathbf{a}_t^*)$. Then we can analyze the accumulated regret up to step T over the context sequence $\{\boldsymbol{\beta}_t\}$, namely, $R_T = \sum_{t=1}^{T} \zeta_t$. Our MAB guarantees a sublinear regret bound for both UCB and TS.

Theorem 3.1. *For $\delta > 0$, take c_t in the UCB as*

$$c_t = 2\log\left(\frac{2^s \pi^2 t^2}{6\delta}\right) \qquad t \in \mathbb{N}$$

where d is the number of PDE parameters, and s is the total number of interface conditions. Conditioning on every context sequence $\{\boldsymbol{\beta}_t\}$, let $\{\mathbf{a}_t\}$ be the action selected by the UCB score under the above choice of $\{c_t\}$. Then, with probability at least $1-\delta$, the regret R_T satisfies

$$R_T \lesssim \sqrt{\frac{2^s T (\log T)^{d+1} \log\left(\frac{2^s T^2}{\delta}\right)}{\log(1+\sigma_0^{-2})}} \quad T = 1, 2, \cdots, \quad (7)$$

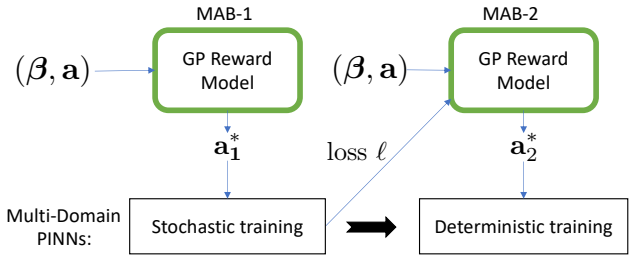


Figure 1: The illustration of the sequential MAB model.

where the implicit constant is absolute (does not depend on $\{c_t\}$ but depends on the domain $\mathcal{X}$). In particular,

$$\mathbb{E}[R_T] \lesssim \sqrt{\frac{2^s T (\log T)^{d+1} \log\left(2^s T^2\right)}{\log(1+\sigma_0^{-2})}}, \quad (8)$$

Moreover, (8) *holds also for Thompson sampling.*

We can see $\lim_{T\to\infty} \frac{\mathbb{E}[R_T]}{T} = 0$, *i.e.*, the average instantaneous regret converges to zero. Since $R_T \geq 0$, it means that our MAB is able to find the optimal interface conditions (almost surely) for every possible sequence of PDE parameters with enough long run.

3.2 Sequential Multi-Arm Bandits

In practice, to achieve good and reliable performance, the training of PINNs is often divided into two stages (Lu et al., 2021). The first phase is stochastic training, typically with ADAM optimizer (Kingma and Ba, 2014), to find a nice valley of the loss landscape. The second phase is deterministic optimization, typically with L-BFGS, to ensure convergence to the (local) minimum. Due to the different nature of the two phases, the ideal interface conditions can vary as well. To enable more flexible choices so as to further improve the performance, we propose a sequential MAB model, as illustrated in Fig. 1. Specifically, for each training phase, we introduce a MAB similar to Sec 3.1, which updates a separate GP reward model. To coordinate the two MAB's and to optimize the final accuracy, the reward of the first MAB, denoted by r_1, includes not only the negative solution error after the stochastic training phase, but also a discounted error after the second phase,

$$r_1 = -\xi_1 + \gamma \cdot (-\xi_2) \quad (9)$$

where γ is the discount factor, and ξ_1 and ξ_2 are the solution errors after the stochastic and deterministic training phases, respectively. In this way, the influence of the interface conditions at the first training phase on the final solution accuracy is also integrated into the learning of the reward model. Next, we expand the context of the second MAB with the training loss value ℓ after the first phase. In this way, the training status of the first phase is also used to determine

Algorithm 1 METALIC-single(T)

Initialize the GP reward model, and $\mathcal{D} \leftarrow \emptyset$.
for $t = 1 \dots T$ **do**
  Randomly sample the PDE parameters $\boldsymbol{\beta} \in \mathcal{X}$.
  For each action $\mathbf{a} \in \mathcal{P}$, compute the predictive distribution of the GP reward model, $\mathcal{N}\left(\mu(\mathbf{a}, \boldsymbol{\beta}), \sigma^2(\mathbf{a}, \boldsymbol{\beta})\right)$.
  Compute the UCB score: $\text{UCB}(\mathbf{a}) = \mu + \sqrt{c_t} \cdot \sigma$ or TS score: $\text{TS}(\mathbf{a}) \sim \mathcal{N}(\mu, \sigma^2)$.
  $\mathbf{a}^* = \text{argmax}_{\mathbf{a} \in \mathcal{P}} \text{UCB}(\mathbf{a})$ or $\mathbf{a}^* = \text{argmax}_{\mathbf{a} \in \mathcal{P}} \text{TS}(\mathbf{a})$.
  Use the interface conditions of $\mathbf{a}^*$ to train the multi-domain PINNs to solve the PDE parameterized by $\boldsymbol{\beta}$, and evaluate the solution error ξ.
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{([\mathbf{a}^*, \boldsymbol{\beta}], -\xi)\}$.
  Retrain the GP reward model on $\mathcal{D}$.
end for

Algorithm 2 METALIC-seq(γ, T)

Initialize two GP reward models. Set their training sets $\mathcal{D}_1$ and $\mathcal{D}_2$ to empty.
repeat
  Randomly sample $\boldsymbol{\beta} \in \mathcal{X}$.
  Based on the predictive distribution of the first GP model, use UCB or TS to select the best action $\mathbf{a}_1^*$.
  Use the interface conditions of $\mathbf{a}_1^*$ to train the multi-domain PINNs with ADAM. Evaluate the error ξ_1 for solving the PDE parameterized by $\boldsymbol{\beta}$.
  Given the current training loss ℓ and $\boldsymbol{\beta}$, compute the predictive distribution of the second GP reward model for each action, and use UCB or TS to select the best action $\mathbf{a}_2^*$.
  Use the interface conditions of $\mathbf{a}_2^*$ to continue the training with L-BFGS. Evaluate the solution error ξ_2.
  $\mathcal{D}_1 \leftarrow \mathcal{D}_1 \cup \{([\mathbf{a}_1^*, \boldsymbol{\beta}], -\xi_1 - \gamma\xi_2)\}, \mathcal{D}_2 \leftarrow \mathcal{D}_2 \cup \{([\mathbf{a}_2^*, \boldsymbol{\beta}, \ell], -\xi_2)\}$.
  Retrain the two GP models on $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively.
until T iterations are done

the interface conditions for the second phase. The learning of the sequential MAB's is summarized in Algorithm 2.

Algorithm Complexity. The time complexity of both MAB algorithms is $\mathcal{O}(TR + \sum_{t=1}^{T} t^3)$ where T is the total number of iterations, R is the complexity of multi-domain PINNs, and $\mathcal{O}(\sum_{t=1}^{T} t^3)$ is the total time complexity of updating the GP reward model to T. In practice, T is typically chosen as a few hundreds (see the experimental section). Under such a choice, running multi-domain PINNs is much more costly than GP training, and the complexity is dominant by $\mathcal{O}(TR)$. Hence, the time complexity is linear in the number of iterations. The space complexity of our algorithm is $\mathcal{O}(C + T^2)$, including the storage of the GP reward model and the multi-domain PINN (with constant complexity $\mathcal{O}(C)$).

4 Related Work

As an alternative to mesh-based numerical methods, PINNs have had many success stories, *e.g.*, (Raissi et al., 2020; Chen et al., 2020; Sirignano and Spiliopoulos, 2018; Zhu et al., 2019; Geneva and Zabaras, 2020; Sahli Costabal et al., 2020). Multi-domain PINNs, *e.g.*, XPINNs (Jagtap and Karniadakis, 2021) and cPINNs (Jagtap et al., 2020), extend PINNs based on domain decomposition and use a set of PINNs to solve the PDE in different subdomains. To stitch together the subdomains, XPINNs used solution continuity and residual continuity as the interface conditions. Other conditions are also available, such as the flux conservation in cPINNs (Jagtap et al., 2020), the gradient continuity (De Ryck et al., 2022), and the residual gradient continuity in gPINNs (Yu et al., 2022). Recently, Hu et al. (2021) developed a theoretical understanding on the convergence and generalization properties of XPINNs, and examined the trade-off between XPINNs and PINNs.

Meta learning (Schmidhuber, 1987; Naik and Mammone, 1992; Thrun and Pratt, 2012) is an important topic in machine learning. The existing works can be roughly attributed to three categories: (1) metric-learning that learns a metric space with which the tasks can make predictions via matching the training points, *e.g.*, nonparametric nearest neighbors (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017; Oreshkin et al., 2018; Allen et al., 2019), (2) learning black-box models (*e.g.*, neural networks) that map the task dataset and hyperparameters to the optimal model parameters or parameter updating rules, *e.g.*, (Andrychowicz et al., 2016; Ravi and Larochelle, 2017; Santoro et al., 2016; Wang et al., 2016; Munkhdalai and Yu, 2017; Mishra et al., 2017), and (3) bi-level optimization where the outer level optimizes the hyperparameters and the inner level optimizes the model parameters given the hyperparameters (Finn et al., 2017; Finn, 2018; Bertinetto et al., 2018; Lee et al., 2019; Zintgraf et al., 2019; Li et al., 2017; Zhou et al., 2018). The bi-level optimization approaches are often restricted by the high cost of computing the meta-gradient (*e.g.*, w.r.t the model initialization) via computational graphs. The issue has been recently addressed by (Li et al., 2023) that uses gradient flows to model the inner-optimization and adjoint state methods to compute the meta-gradients. Recently, Penwarden et al. (2023) developed the first method to meta learn the initialization for PINNs.

Multi-arm bandit is a classical online decision making framework (Lai et al., 1985; Auer et al., 2002a;b; Mahajan and Teneketzis, 2008; Bubeck et al., 2012), and have numerous applications, such as online advertising (Avadhanula et al., 2021), collaborative filtering (Li et al., 2016), clinical trials (Aziz et al., 2021) and robot control (Laskey et al., 2015). To our knowledge, our work is the first to use MAB for meta learning of task-specific hyperparameters, which is advantageous in its simplicity and efficiency. MAB can be viewed as an instance of reinforcement learning (RL) (Sutton and Barto, 2018), but it only needs to estimate a reward function online. While one can design more expressive RL models to meanwhile learn a Markov decision process (in

MAB, we simply use UCB or TS), it often demands we run a massive number of PINN training trajectories, which is much more expensive. The model estimation is also much more challenging.

5 Experiment

To evaluate METALIC, we considered four commonly-used benchmark PDE families in PINN literature (Raissi et al., 2019; Krishnapriyan et al., 2021; Jagtap and Karniadakis, 2021). Note that our work is *not* about how to smartly decompose the domain or purposely decompose with strong physical meanings. We followed (Jagtap and Karniadakis, 2021) to use a naive decomposition to verify the effectiveness of our interface condition selection method.

The Poisson Equation. First, we considered a 2D Poisson equation with a parameterized source function,

$$u_{xx} + u_{yy} = \widetilde{h}(x, y; s) \tag{10}$$

where $(x, y) \in [0, 1] \times [0, 1]$, $\widetilde{h}(x, y; s) = h(x, y; s)/\max_{x,y} h(x, y; s)$, and

$$h(x, y; s) = [\text{erf}((x - 0.25)s) - \text{erf}((x - 0.75)s)] \cdot [\text{erf}((y - 0.25)s) - \text{erf}((y - 0.75)s)], \tag{11}$$

where $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \mathrm{d}t$, and $s \in [0, 50]$ is the sharpness parameter that controls the sharpness of the interior square in the source. We used Dirichlet boundary conditions, and ran a finite difference solver to obtain an accurate "gold-standard" solution. To run multi-domain PINNs, we split the domain into two subdomains, where the interface is a line at $y = 0.5$. We visualize an exemplar solution and the subdomains, including the sampled boundary and collocation points in Fig. 5 of Appendix.

Advection Equation. We next considered a 1D advection (one-way wave) equation,

$$u_t + \beta u_x = 0$$

where $x \in [0, 2\pi]$, $t \in [0, 1]$, and β is the PDE parameter denoting the wave speed. We used Dirichlet boundary conditions, and the solution has an analytical form, $u(x, t) = q(x - \beta t)$ were $q(x)$ is the initial condition (which we selected as $q(x) = \sin(x)$). For domain decomposition, we split the domain at $t = 0.5$ to obtain two subdomains. Fig. 6 in Appendix shows an exemplar solution and the subdomains with the interface.

Reaction Equation. Third, we evaluated a 1D reaction equation,

$$u_t - \rho u(1 - u) = 0$$

where ρ is the reaction coefficient (ODE parameter), $x \in [0, 2\pi]$, $t \in [0, 1]$ and $u(x, 0) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}}$. The exact solution is $u(x, t) = u(x, 0) \cdot [e^{\rho t} / (u(x, 0)e^{\rho t} + 1 - u(x, 0))]$. We split the domain at $t = 0.5$ to obtain two subdomains. Although not required for well-posedness of the ODE system, because we are solving for the PINN space-time field $u(x, t)$, we use the exact solution to define a boundary loss term. This enhances training without compromising the time partitioning we wish to highlight. We show a solution example and the subdomains in Fig. 7 of Appendix.

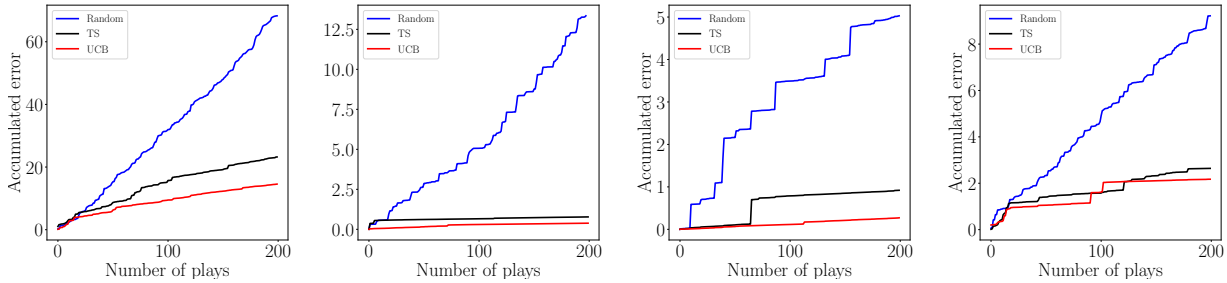
Burger's Equation. Fourth, we considered the viscous Burger's equation,

$$u_t + uu_x = \nu u_{xx}$$

where $\nu \in [0.001, 0.05]$ is the viscosity (PDE parameter), $x \in [-1, 1]$, $t \in [0, 1]$, and $u(x, 0) = -\sin(\pi x)$. We ran a numerical solver to obtain an accurate "gold-standard" solution. To decompose the domain, we take the middle portion that includes the shock waves as one subdomain, namely, $\Omega_1 : x \in [-0.1, 0.1], t \in [0, 1]$, and the remaining as the other subdomain, $\Omega_2 : x \in [-1, -0.1] \cap [0.1, 1], t \in [0, 1]$. Hence, the interface consists of two lines. See Fig. 8 in Appendix for the illustration and solution example.

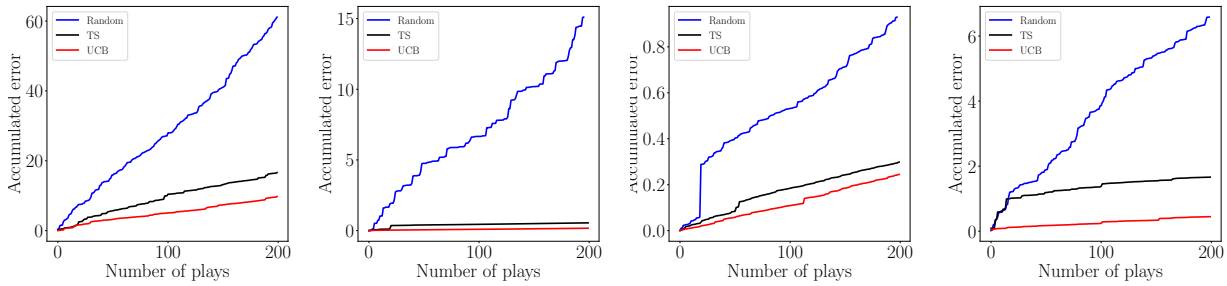
To evaluate METALIC, we used nine interface conditions, which are listed in Appendix (Sec. A). For the PINN in each subdomain, we used two layers, with 20 neurons per layer and tanh activation function. We randomly sampled 1,000 collocation points and 100 boundary points for each PINN. To inject the interface conditions, we randomly sampled 101 interface points for the Poisson, advection and reaction equations, and 802 interface points for Burger's equation. We set $\lambda_b = 20$ and $\lambda_I = 5$, which follows the insight of (Wang et al., 2021; 2022) to adopt large weights for the boundary and interface terms so as to prevent the training of PINNs from being dominated by the residual term. We denote our single MAB by METALIC-single, and sequential MABs by METALIC-seq. For the latter, we set the discount factor $\gamma = 0.9$ (see (9)). For better numerical stability, we used the relative L_2 error in the log domain to obtain the reward for updating the GP models. The running of the multi-domain PINNs consists of 10K ADAM epochs (with learning rate 10^{-3}) and then 50K L-BFGS iterations (the first order optimality and parameter change tolerances set to 10^{-6} and 10^{-9} respectively). We set $c_t = 1$ to compute the UCB score. We ran 200 plays (iterations) for our method. For static (offline) test, we randomly sampled 100 PDEs (which do not overlap with the PDEs sampled during the online playing). We then used the learned reward model to determine the best interface conditions for each particular PDE (according to the predictive mean), with which we ran the multi-domain PINNs to solve the PDE, and computed the relative L_2 error.

First, to examine the online performance of METALIC, we looked into the accumulated solution error along with the number of plays. We compared with randomly selecting the



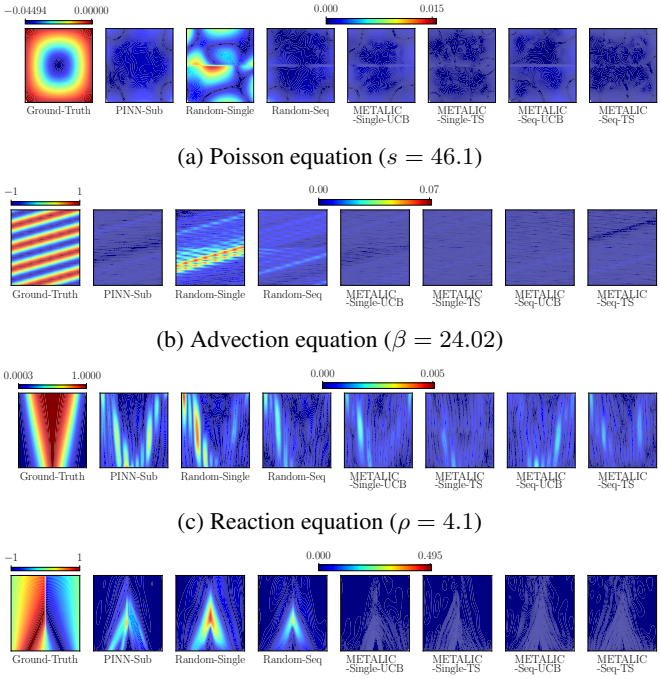
(a) Poisson equation (b) Advection equation (c) Reaction equation (d) Burgers' equation

Figure 2: Online performance of METALIC-Single.



(a) Poisson equation (b) Advection equation (c) Reaction equation (d) Burgers' equation

Figure 3: Online performance of METALIC-Seq.



(a) Poisson equation ($s = 46.1$)

(b) Advection equation ($\beta = 24.02$)

(c) Reaction equation ($\rho = 4.1$)

(d) Burgers' equation ($\nu = 0.0036$)

Figure 4: Point-wise solution error.

arm at each play. In the case of running METALIC-seq, this baseline correspondingly randomly selects the arm twice, one at the stochastic training phase, and the other at the deterministic phase. The results are shown in Fig. 2 and 3. As we can see, the accumulated error of METALIC with both UCB and TS grows much slower, *i.e.*, sublinearly, than the random selection approach (note that the reward of the optimal action is unknown due to the randomness in the running of PINNs, and we cannot compute the regret). This has shown that our method achieves a much better exploration-exploitation tradeoff in the online interface condition decision and model updating, which is consist with many other MAB applications (see Sec 4). The results demonstrate the advantage of our MAB-based approach. First, via effective exploration, METALIC can collect valuable training examples (rewards at new actions and context) to improve the learning efficiency and performance of the GP reward model. Second, the online decision also takes advantage of the predictive ability of the current reward model, *i.e.*, exploitation, to select effective interface conditions, which results in increasingly better solution accuracy of the multi-domain PINNs. The online nature of METALIC enables us to keep improving the reward model while utilizing it to solve new equations with promising accuracy.

Next, we conducted the offline test, namely, without on-

Method	Poisson	Advection	Reaction	Burger's
Random-Single	0.3992 ± 0.0037	0.07042 ± 0.01575	0.00612 ± 0.000278	0.04021 ± 0.00057
Random-Seq	0.3004 ± 0.0032	0.03922 ± 0.00919	0.01284 ± 0.000707	0.03486 ± 0.00066
PINN-Sub	0.03078 ± 0.00177	$0.00130 \pm 8.108\text{e-}5$	0.00213 ± 0.00021	0.00738 ± 0.00313
PINN-Merge-H	0.02398 ± 0.00144	$0.00098 \pm 5.038\text{e-}5$	0.00223 ± 0.00026	0.00951 ± 0.00390
PINN-Merge-V	0.02184 ± 0.00211	$0.00079 \pm 3.391\text{e-}5$	$\mathbf{0.00099 \pm 0.00013}$	0.00276 ± 0.00041
METALIC-Single-TS	0.02503 ± 0.0002	$0.00079 \pm 4.5897\text{e-}5$	$0.00204 \pm 1.013\text{e-}4$	$\mathbf{0.00109 \pm 1.306\text{e-}5}$
METALIC-Single-UCB	0.0245 ± 0.0002	$0.00078 \pm 3.6771\text{e-}5$	$0.00102 \pm 8.945\text{e-}6$	$0.00161 \pm 2.939\text{e-}5$
METALIC-Seq-TS	$0.01639 \pm 9.5384\text{e-}5$	$0.00078 \pm 3.6473\text{e-}5$	$\mathbf{0.00099 \pm 8.4704\text{e-}6}$	$0.00152 \pm 5.571\text{e-}5$
METALIC-Seq-UCB	$\mathbf{0.01406 \pm 9.1099\text{e-}5}$	$\mathbf{0.00070 \pm 3.2790\text{e-}5}$	$\mathbf{0.00099 \pm 5.999\text{e-}6}$	$0.00139 \pm 3.948\text{e-}5$

Table 1: The average L_2 relative error of single-domain PINNs and multi-domain PINNs for solving 100 test PDEs. The interface conditions of the multi-domain PINNs are provided by METALIC and random selection. {Single, Seq} indicate using a single set or two sequential sets of interface conditions for the running of the multi-domain PINNs. {TS, UCB} corresponds to our method using TS or UCB score to determine the interface conditions at each play.

Methods	Error Reduction % by METALIC			
	Poisson's	*Advection*	*Reaction*	*Burgers'*
PINN-Sub	54.3	46.2	53.5	85.2
PINN-Merge-H	41.4	28.6	55.6	88.5
PINN-Merge-V	35.6	11.4	0	60.5

Table 2: Percentage of error reduction by METALIC as compared with single domain PINNs.

line exploration and model updating any more after 200 plays. We compared with (1) Random-Single, which, for each PDE, randomly selects a set of interface conditions applied to the entire training of the multi-domain PINNs, and (2) Random-Seq, which for each PDE, randomly selects two sets of interface conditions, one for the stochastic training and the other for the deterministic training phase. We also tested single-domain PINNs that do not incorporate interface conditions. Specifically, we compared with (3) PINN-Sub, which used the same architecture as the PINN in each subdomain, but is applied to the entire domain, (4) PINN-Merge-H, which horizontally pieced all the PINNs in the subdomains, *i.e.*, doubling the layer width yet fixing the depth, (5) PINN-Merge-V, which vertically stacked the PINNs, *i.e.*, doubling the depth while fixing the width. *While PINN-Merge-H and PINN-Merge-V merge the PINNs in each subdomain, the total number of neurons actually increases (for connecting these PINNs). Hence, the merged PINN is more expressive.* Each single-domain PINN used the union of the boundary points and collocation points from every subdomain. We used the same weight for the boundary term, *i.e.*, $\lambda_b = 20$. The optimization of each single-domain PINN follows the same setting of the multi-domain PINNs (*i.e.*, 10K ADAM epochs and 50K L-BFGS iterations).

We report the average relative L_2 solution error and the standard deviation in Table 1. As we can see, randomly selecting interface conditions, no matter for the whole training procedure or two training phases, result in much worse solution accuracy of multi-domain PINNs. The solution error is one order of magnitude bigger than METALIC in all the settings. It confirms that the success of the multi-domain PINNs is up to appropriate interface conditions. Next, we can observe that while the performance of METALIC-Single is similar to METALIC-Seq, the best solution accuracy is in most cases obtained by interface conditions selected by METALIC-Seq (except in solving the Burger's equation). It demonstrates that our sequential MAB model that can employ different conditions for the two training phases is more flexible and brings additional improvement. We also observe that in most cases using UCB for online playing can lead to better performance for both METALIC-Single and METALIC-Seq. This is consistent with the online performance evaluation (see Fig. 2 and 3). Third, among the single-domain PINN methods, PINN-Merge-V outperforms PINN-sub in all the equation families and PINN-Merge-H outperforms PINN-sub in the Poisson and advection equations, showing that deeper or wider architectures can help further improve the solution accuracy. However, their performance is still second to the best setting of METALIC, which uses simpler subdomain PINN architectures and fewer total learnable parameters. These multi-domain PINNs can be further parallelized to accelerate training. By contrast, if the interface conditions are inferior, such as those selected by Random-Single and Random-Seq, the solution error becomes much worse (orders of magnitude bigger) than single-domain PINNs.

In Table 2, we show the percentage of the error reduction led by METALIC (the best setting) , as compared with single-domain PINNs. We can see that, METALIC can give a large reduction in all the cases, except for *Reaction* equation, PINN-Merge-V achieves the same error. Note that both PINN-Merge-V and PINN-Merge-H include more NN parameters than multi-domain PINNs. Together these have shown the importance of the interface conditions for

multi-domain PINNs and the advantage of our method.

For a fine-grained comparison, we visualize the point-wise solution error of PINN-Sub, Random-Single, Random-Seq, and our method in solving four random instances of the equations. As shown in Fig. 4, the point-wise error of both METALIC-Single and METALIC-Seq is quite uniform across the domain and close to zero (dark blue). By contrast, the competing methods often exhibit relatively large errors in a few local regions, *e.g.*, those in the middle (where the shock waves appear) of the domain of Burger's equation (PINN-Sub, Random-Single, Random-Seq), and the central part of the domain of the Poisson and advection equation (Random-Single). It shows that our method not only can give a superior global accuracy, but locally also better recovers individual solution values.

Analysis of Selected Conditions. Finally, we analyzed the selected interface conditions in each equation family by METALIC. We found that those conditions are interesting in that they are physically meaningful, consistent with the properties of the equations, and also tied to the specific optimization step in the multi-domain PINN training. Due to the space limit, we provide the details in Sec. D of Appendix.

6 Discussion

One might be concerned about the curse of dimensionality issue. That is, when there is a large number of subdomains and we still learn different conditions for every interface, a great many MAB's are needed and the online learning cost can explode. Here we argue that, while theoretically possible, such issue in practical usage does not occur and we do not need to worry about it.

In general, we have two motivations to apply domain decomposition. The first motivation is to improve the solution accuracy. In this case, one prefers to adopt different conditions across different interfaces. However, due to the rich expressivity of neural networks, *e.g.*, universal approximation, PINNs are like high-order finite element methods, and the number of subdomains should be as few as possible. For example, the recent work (Jagtap et al., 2022) uses multi-domain PINNs for inverse problems in supersonic flows (Euler equations), which largely outperforms the single domain PINNs. Only two interfaces and three subdomains are used in (Jagtap et al., 2022) for this problem (see Fig. 3, 4, and 5 of that paper). Hence, to learn the interface conditions, we only need a few more MAB's and the growth of the learning cost is minor.

The second motivation of domain decomposition is to enable parallel computation so as to speed up problem solving. In such case, one often prefers a large number of subdomains, and each may be allocated to a separate computing unit, *e.g.*, a CPU core. However, to ensure the solution is consistent (not influenced by the number of computing units), one needs to demand the interface conditions be identical across all the subdomains. That means, METALIC only needs one MAB (or two sequential MAB's; see Fig. 1) to learn the interface conditions shared by all the subdomains. Hence, there is again no "curse of dimensionality" issue. Note that the parallel computational framework for multi-domain PINNs have been published (Shukla et al., 2021).

7 Conclusion

We have presented METALIC, a simple, efficient and powerful meta learning approach to select PDE-specific interface conditions for general multi-domain PINNs. The results at four bench-mark equation families are encouraging. In the future, we will use the PDE residual as the approximate reward so that our method can be fully unsupervised. We will also extend METALIC to meta learn the interface locations along with the conditions, as a function of not only accuracy but training time so as to improve both the solution accuracy and training efficiency of the multi-domain PINNs.

Acknowledgments

This work has been supported by MURI AFOSR grant FA9550-20-1-0358, NSF CAREER Award IIS-2046295, and NSF DMS-1848508.

References

Allen, K., Shelhamer, E., Shin, H., and Tenenbaum, J. (2019). Infinite mixture prototypes for few-shot learning. In International Conference on Machine Learning, pages 232–241. PMLR.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. arXiv preprint arXiv:1606.04474.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. Journal of Machine Learning Research, 3(Nov):397–422.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. SIAM journal on computing, 32(1):48–77.

Avadhanula, V., Colini Baldeschi, R., Leonardi, S., Sankararaman, K. A., and Schrijvers, O. (2021). Stochastic bandits for multi-platform budget optimization in on-

line advertising. In Proceedings of the Web Conference 2021, pages 2805–2817.

Aziz, M., Kaufmann, E., and Riviere, M.-K. (2021). On multi-armed bandit designs for dose-finding clinical trials. Journal of Machine Learning Research, 22(1-38):4.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2018). Meta-learning with differentiable closed-form solvers. arXiv preprint arXiv:1805.08136.

Bogunovic, I. and Krause, A. (2021). Misspecified gaussian process bandit optimization. Advances in Neural Information Processing Systems, 34:3004–3015.

Bubeck, S., Cesa-Bianchi, N., et al. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Foundations and Trends® in Machine Learning, 5(1):1–122.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In Advances in neural information processing systems, pages 2249–2257.

Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. (2020). Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express, 28(8):11618–11633.

De Ryck, T., Jagtap, A. D., and Mishra, S. (2022). Error estimates for physics informed neural networks approximating the navier-stokes equations. arXiv preprint arXiv:2203.09346.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779.

Finn, C. (2018). Learning to learn with gradients. PhD thesis, UC Berkeley.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning, pages 1126–1135. PMLR.

Geneva, N. and Zabaras, N. (2020). Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics, 403:109056.

Hochreiter, S., Younger, A. S., and Conwell, P. R. (2001). Learning to learn using gradient descent. In International Conference on Artificial Neural Networks, pages 87–94. Springer.

Hu, Z., Jagtap, A. D., Karniadakis, G. E., and Kawaguchi, K. (2021). When do extended physics-informed neural networks (xpinns) improve generalization? arXiv preprint arXiv:2109.09444.

Jagtap, A. D. and Karniadakis, G. E. (2021). Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In AAAI Spring Symposium: MLPS.

Jagtap, A. D., Kharazmi, E., and Karniadakis, G. E. (2020). Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Computer Methods in Applied Mechanics and Engineering, 365:113028.

Jagtap, A. D., Mao, Z., Adams, N., and Karniadakis, G. E. (2022). Physics-informed neural networks for inverse problems in supersonic flows. Journal of Computational Physics, 466:111402.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kissas, G., Yang, Y., Hwuang, E., Witschey, W. R., Detre, J. A., and Perdikaris, P. (2020). Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering, 358:112623.

Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In ICML deep learning workshop, volume 2. Lille.

Krause, A. and Ong, C. (2011). Contextual gaussian process bandit optimization. Advances in neural information processing systems, 24.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34:26548–26560.

Lai, T. L., Robbins, H., et al. (1985). Asymptotically efficient adaptive allocation rules. Advances in applied mathematics, 6(1):4–22.

Laskey, M., Mahler, J., McCarthy, Z., Pokorny, F. T., Patil, S., van den Berg, J., Kragic, D., Abbeel, P., and Goldberg, K. (2015). Multi-armed bandit models for 2d grasp planning with uncertainty. In 2015 IEEE International Conference on Automation Science and Engineering (CASE), pages 572–579.

Lattimore, T. and Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10657–10665.

Li, S., Karatzoglou, A., and Gentile, C. (2016). Collaborative filtering bandits. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, pages 539–548.

Li, S., Wang, Z., Narayan, A., Kirby, R., and Zhe, S. (2023). Meta-learning with adjoint methods. In International Conference on Artificial Intelligence and Statistics, pages 7239–7251. PMLR.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. arXiv preprint arXiv:1707.09835.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. (2021). Deepxde: A deep learning library for solving differential equations. SIAM Review, 63(1):208–228.

Mahajan, A. and Teneketzis, D. (2008). Multi-armed bandit problems. In Foundations and applications of sensor management, pages 121–151. Springer.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2017). A simple neural attentive meta-learner. arXiv preprint arXiv:1707.03141.

Munkhdalai, T. and Yu, H. (2017). Meta networks. In International Conference on Machine Learning, pages 2554–2563. PMLR.

Naik, D. K. and Mammone, R. J. (1992). Meta-neural networks that learn by learning. In [Proceedings 1992] IJCNN International Joint Conference on Neural Networks, volume 1, pages 437–442. IEEE.

Oreshkin, B. N., Rodriguez, P., and Lacoste, A. (2018). Tadam: Task dependent adaptive metric for improved few-shot learning. arXiv preprint arXiv:1805.10123.

Penwarden, M., Zhe, S., Narayan, A., and Kirby, R. M. (2023). A metalearning approach for physics-informed neural networks (PINNs): Application to parameterized PDEs. Journal of Computational Physics, page 111912.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). 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.

Raissi, M., Yazdani, A., and Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In In International Conference on Learning Representations (ICLR).

Sahli Costabal, F., Yang, Y., Perdikaris, P., Hurtado, D. E., and Kuhl, E. (2020). Physics-informed neural networks for cardiac activation mapping. Frontiers in Physics, 8:42.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In International conference on machine learning, pages 1842–1850. PMLR.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. PhD thesis, Technische Universität München.

Shukla, K., Jagtap, A. D., and Karniadakis, G. E. (2021). Parallel physics-informed neural networks via domain decomposition. Journal of Computational Physics, 447:110683.

Sirignano, J. and Spiliopoulos, K. (2018). Dgm: A deep learning algorithm for solving partial differential equations. Journal of computational physics, 375:1339–1364.

Slivkins, A. et al. (2019). Introduction to multi-armed bandits. Foundations and Trends® in Machine Learning, 12(1-2):1–286.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. Advances in neural information processing systems, 30.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In Proceedings of the 27th International Conference on International Conference on Machine Learning, pages 1015–1022.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. arXiv preprint arXiv:0912.3995.

Sun, L., Gao, H., Pan, S., and Wang, J.-X. (2020). Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Computer Methods in Applied Mechanics and Engineering, 361:112732.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika, 25(3/4):285–294.

Thrun, S. and Pratt, L. (2012). Learning to learn. Springer Science & Business Media.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. arXiv preprint arXiv:1606.04080.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. arXiv preprint arXiv:1611.05763.

Wang, S., Teng, Y., and Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5):A3055–A3081.

Wang, S., Yu, X., and Perdikaris, P. (2022). When and why pinns fail to train: A neural tangent kernel perspective. Journal of Computational Physics, 449:110768.

Yu, J., Lu, L., Meng, X., and Karniadakis, G. E. (2022). Gradient-enhanced physics-informed neural networks for forward and inverse pde problems. Computer Methods in Applied Mechanics and Engineering, 393:114823.

Zhou, F., Wu, B., and Li, Z. (2018). Deep meta-learning: Learning to learn in the concept space. arXiv preprint arXiv:1802.03596.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394:56–81.

Zintgraf, L., Shiarli, K., Kurin, V., Hofmann, K., and Whiteson, S. (2019). Fast context adaptation via meta-learning. In International Conference on Machine Learning, pages 7693–7702. PMLR.

Appendix

A Interface Conditions

We used a total number of 9 interface conditions throughout all the experiments, which are listed in Table 1. Note that I_z and I_{zz} correspond to the first and second-order derivatives w.r.t an input to the PDE solution function. Since all the test PDE problems consist of two spatial or spatiotemporal dimensions, I_z and I_{zz} give four interface conditions. There are no mixed derivatives across different input dimensions. In the case that one I_z is the same as I_c, such as in Poisson equation, the de-duplication gives 9 different conditions. In the case that all I_z's are different from I_c, such as in Burger's equation, we used I_c and removed one I_z ($z = y$ or $z = t$), so that we still maintain 9 interface conditions to be consistent with other experiments.

Table 3: Interface Conditions of Multi-domain PINNs

I_u	Solution continuity (12)
$I_{u_{avg}}$	Average solution continuity (13)
I_r	Residual (14)
I_{rc}	Residual continuity (15)
I_{gr}	Gradient-enhanced residual (16)
I_c	Flux continuity (17)
I_z	First-order spatial/temporal derivative continuity (18)
I_{zz}	Second-order spatial/temporal derivative continuity (19)

$$I_u(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'}) - \widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}^i_{k,k'})\right)^2 \tag{12}$$

$$I_{u_{avg}}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'}) - \widehat{u}^{\text{avg}}_{k,k'}(\mathbf{x}^i_{k,k'})\right)^2 \tag{13}$$

$$\text{where } \widehat{u}^{\text{avg}}_{k,k'}(\mathbf{x}^i_{k,k'}) = \frac{1}{2}\left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'}) + \widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}^i_{k,k'})\right)$$

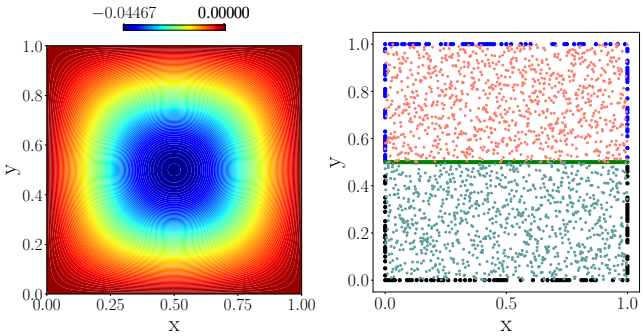
$$I_r(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_k}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right)^2 + \left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_{k'}}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right)^2\right) \tag{14}$$

$$I_{rc}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_k}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right) - \left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_{k'}}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right)\right)^2 \tag{15}$$

$$I_{gr}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \sum_{j=1}^{2} \left(\left|\frac{\partial}{\partial \mathbf{x}^i_{k,k'}[j]}\left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_k}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right)\right|^2 + \left|\frac{\partial}{\partial \mathbf{x}^i_{k,k'}[j]}\left(\mathcal{F}[\widehat{u}_{\boldsymbol{\theta}_{k'}}](\mathbf{x}^i_{k,k'}) - f(\mathbf{x}^i_{k,k'})\right)\right|^2\right) \tag{16}$$

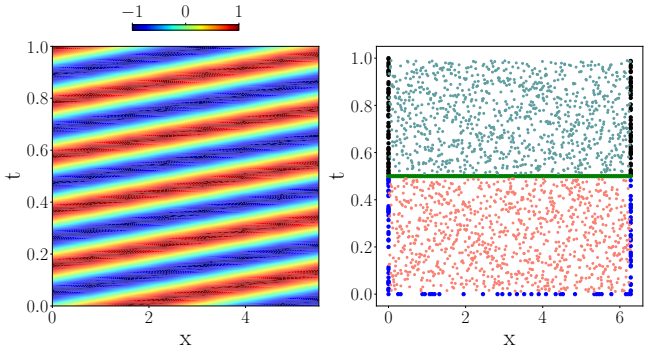
$$I_c(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\phi\left(\widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'})\right) \cdot \mathbf{n} - \phi\left(\widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}^i_{k,k'})\right) \cdot \mathbf{n}\right)^2 \tag{17}$$

$$\text{where } \phi(\widehat{u}_{\boldsymbol{\theta}}) \cdot \mathbf{n} \text{ are fluxes normal at the interface}$$



(a) Solution at $s = 20$ (b) Subdomains

Figure 5: The Poisson equation. The interface is the green line at $y = 0.5$. Blue and black dots show the sampled boundary points in each subdomain, and the internal dots (red and cyan) are the sampled collocation points inside each subdomain.

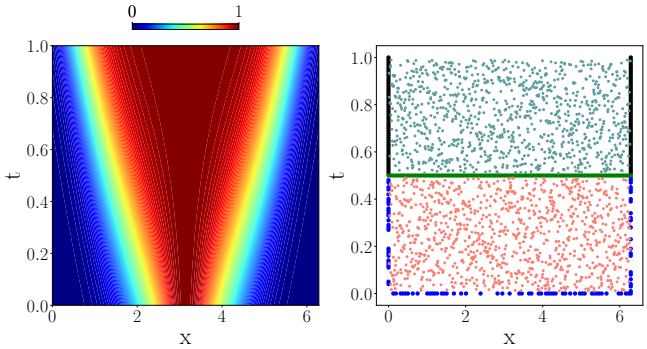


(a) Solution at $\beta = 30$ (b) Subdomains

Figure 6: Advection equation. The interface is the green line at $t = 0.5$.

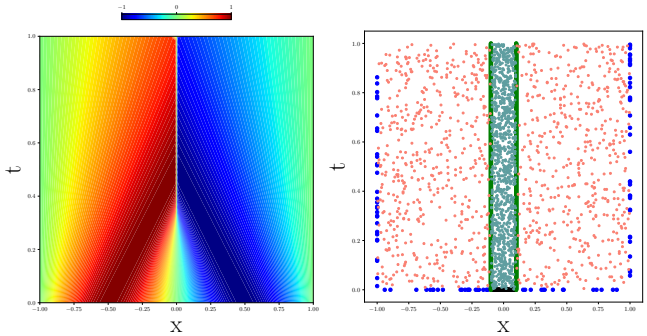
$$I_z(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\frac{\partial}{\partial z^i} \widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'}) - \frac{\partial}{\partial z^i} \widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}^i_{k,k'}) \right)^2 \quad \text{where } z^i = \mathbf{x}^i_{k,k'}[1] \text{ or } z^i = \mathbf{x}^i_{k,k'}[2]. \tag{18}$$

$$I_{zz}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k'}) = \frac{1}{J_{k,k'}} \sum_{i=1}^{J_{k,k'}} \left(\frac{\partial^2}{\partial {z^i}^2} \widehat{u}_{\boldsymbol{\theta}_k}(\mathbf{x}^i_{k,k'}) - \frac{\partial^2}{\partial {z^i}^2} \widehat{u}_{\boldsymbol{\theta}_{k'}}(\mathbf{x}^i_{k,k'}) \right)^2 \quad \text{where } z^i = \mathbf{x}^i_{k,k'}[1] \text{ or } z^i = \mathbf{x}^i_{k,k'}[2]. \tag{19}$$



(a) Solution at $\rho = 5.0$ (b) Subdomains

Figure 7: Reaction equation. The interface is at $t = 0.5$.



(a) Solution at $\nu = 0.001$ (b) Subdomains

Figure 8: Burger's equation. The interfaces are at $x = -0.1$ and $x = 0.1$. The middle portion (filled with cyan dots) is the first subdomain, and the remaining parts constitute the second subdomain.

B Regret Bound Proof

Recall that $\mathcal{X} \subseteq \mathbb{R}^d$ is a compact set denoting the parameter (context) space associated with the parametrized PDE, and $\mathcal{S}$ is the state space consisting of a finite number of interface conditions, i.e. $|\mathcal{S}| = s \in \mathbb{N}$. The action space $\mathcal{P}$ is defined as

$$\mathcal{P} = 2^{\mathcal{S}} = \{(q_1, \cdots, q_s) \in \{0,1\}^s\} \subset \mathbb{R}^s.$$

In our paper, the reward at time t is modeled as

$$r_t = r(\boldsymbol{\beta}_t, a_t) = f(\boldsymbol{\beta}_t, a_t) + \eta_t \qquad (\boldsymbol{\beta}_t, a_t) \in \mathcal{X} \times \mathcal{P}, \tag{20}$$

where $\boldsymbol{\beta}_t$ is the context revealed at time t, a_t is the selected action, f is a function on $\mathcal{X} \times \mathcal{P}$ sampled from an appropriate prior, and η_t is a white noise process used to model the extraneous randomness (e.g. neural network implementation, error rounding, etc.)

In our case, the true reward is the negative error metric computed for the learned PDE solution, which is too complicated for analysis. Alternatively, we use the above model (20) as a substitute for approximation. As a result, the reward model is misspecified. Nevertheless, we assume that (20) is reflective of the true reward and do not consider the model misspecification effects in the subsequent analysis for ease of demonstration; ideas from (Bogunovic and Krause, 2021) can be used to obtain refined analysis for misspecified models but we do not pursue them here.

We now state the technical assumptions on the model parametrization as used in the METALIC algorithm:

- $f(\boldsymbol{\beta}, a)$ is sampled from a $\mathcal{GP}(0, \kappa)$ prior, where κ is a kernel function on $\mathcal{X} \times \mathcal{P} \subset \mathbb{R}^{d+s}$:

$$\kappa((\boldsymbol{\beta}, a), (\boldsymbol{\beta}', a')) = \kappa_1(\boldsymbol{\beta}, \boldsymbol{\beta}')\kappa_2(a, a'),$$

  where κ_1 and κ_2 are Gaussian kernels:

$$\kappa_1(\boldsymbol{\beta}, \boldsymbol{\beta}') = \exp(-\tau_1 \|\boldsymbol{\beta} - \boldsymbol{\beta}'\|_2^2) \qquad \kappa_2(a, a') = \exp\left(-\frac{\tau_2}{s}\|a - a'\|_1\right).$$

  (Note: The key assumption we will be using the is the tensor product structure as well as the form of κ_1; since κ_2 is discrete, it does not encode much of geometry and changing to other alternatives should not affect the subsequent analysis.)

- η_t are i.i.d. Gaussian with variance σ_0^2:

$$\eta_t \overset{\text{i.i.d.}}{\sim} N(0, \sigma_0^2).$$

Under the above assumptions, for $T \in \mathbb{N}$ and historical observations $\mathbf{x}_t = (\boldsymbol{\beta}_t, a_t) \in \mathcal{X} \times \mathcal{P}$, $1 \le t \le T$, $y_T = (r_1, \cdots, r_T)^\top$, the posterior distribution of f at $\mathbf{x} = (\boldsymbol{\beta}, a) \in \mathbb{R}^{d+s}$ is a normal random variable with mean and variance given below:

$$\begin{aligned}
\mu_T(\mathbf{x}) &= k_T^\top(\mathbf{x})(\sigma_0^2 I_T + K_T)^{-1} y_T \\
\sigma_T^2(\mathbf{x}) &= \kappa(\mathbf{x}, \mathbf{x}) - k_T^\top(\mathbf{x})(\sigma_0^2 I_T + K_T)^{-1} k_T(\mathbf{x}),
\end{aligned}$$

where

$$K_T = (\kappa(\mathbf{x}_i, \mathbf{x}_j))_{1 \le i,j \le T} \in \mathbb{R}^{T \times T} \qquad k_T(\mathbf{x}) = (\kappa(\mathbf{x}, \mathbf{x}_1), \cdots, \kappa(\mathbf{x}, \mathbf{x}_T))^\top \in \mathbb{R}^T.$$

The following quantity, which measures the maximum uncertainty reduction of $f_T = (f(\mathbf{x}_1), \cdots, f(\mathbf{x}_T))^\top$ when observing y_T, will appear in the regret analysis:

$$\begin{aligned}
\gamma_T &:= \max_{\{\mathbf{x}_t\} \subset \mathcal{X} \times \mathcal{P}} H(f_T) - H(f_T | y_T) \\
&= \max_{\{\mathbf{x}_t\} \subset \mathcal{X} \times \mathcal{P}} H(y_T) - H(y_T | f_T) \qquad H(y_T|f_T) \text{ is independent of } \{\mathbf{x}_t\} \\
&= \max_{\{\mathbf{x}_t\} \subset \mathcal{X} \times \mathcal{P}} \frac{1}{2} \log |I_T + \sigma_0^{-2} K_T|,
\end{aligned}$$

where H is the Shannon entropy.

We are now ready to state the main result:

Theorem B.1. *For $\delta > 0$, take c_t in the UCB algorithm as*

$$c_t = 2 \log\left(\frac{2^s \pi^2 t^2}{6\delta}\right) \qquad t \in \mathbb{N}.$$

Conditioning on every context sequence $\{\boldsymbol{\beta}_t\}$, let $\{a_t\}$ be the action selected by the UCB algorithm under the above choice of $\{c_t\}$. Then, with probability at least $1 - \delta$, the regret R_T satisfies

$$R_T \lesssim \sqrt{\frac{2^s T (\log T)^{d+1} \log\left(\frac{2^s T^2}{\delta}\right)}{\log(1 + \sigma_0^{-2})}} \qquad T = 1, 2, \cdots, \tag{21}$$

where the implicit constant is absolute (does not depend on $\{\boldsymbol{\beta}_t\}$ but depends on the domain $\mathcal{X}$). In particular,

$$\mathbb{E}[R_T] \lesssim \sqrt{\frac{2^s T (\log T)^{d+1} \log(2^s T^2)}{\log(1 + \sigma_0^{-2})}}, \tag{22}$$

Moreover, (22) *holds also for the Thompson sampling algorithm.*

Proof. We first prove the statement concerning the UCB algorithm. The proof is similar to (Krause and Ong, 2011) overall. Owing to a few subtle differences, we provide a sketch of the proof. In our setting, contexts are revealed in a random fashion that is independent of the reward noises. For convenience, we condition on the context sequence $\{\boldsymbol{\beta}_t\}_{t\in\mathbb{N}} \subset \mathcal{X}$ throughout the analysis, i.e., we treat $\{\boldsymbol{\beta}_t\}_{t\in\mathbb{N}} \subset \mathcal{X}$ as deterministic and arbitrary sequence.

Firstly, a standard application of concentration inequalities and a union bound (Krause and Ong, 2011, supplement, Lemma 5.2) yields that, with probability at least $1-\delta$,

$$|f(\boldsymbol{\beta}_t, a) - \mu_t(\boldsymbol{\beta}_t, a)| \le c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a) \qquad t \in \mathbb{N},\ a \in \mathcal{P}, \tag{23}$$

which immediately implies an upper bound for the regret:

$$\begin{aligned}
R_T &= \sum_{t=1}^{T}(f(\boldsymbol{\beta}_t, a_t^*) - f(\boldsymbol{\beta}_t, a_t)) \qquad a_t^* = \arg\max_{a\in\mathcal{P}} f(\boldsymbol{\beta}_t, a) \\
&\le \sum_{t=1}^{T} \underbrace{(\mu_{t-1}(\boldsymbol{\beta}_t, a_t^*) + c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t^*)) - (\mu_{t-1}(\boldsymbol{\beta}_t, a_t) + c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t))}_{\le 0} + 2c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t) \\
&\le 2\sum_{t=1}^{T} c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t) \\
&\le 2\sqrt{T}\left[c_T \sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{\beta}_t, a_t)\right]^{1/2} \qquad \text{(Cauchy–Schwarz; } c_t \text{ is increasing in } t\text{)} \\
&\overset{(\star)}{\le} \sqrt{\frac{8Tc_T\gamma_T}{\log(1+\sigma_0^{-2})}} \\
&\lesssim K(\sigma_0)\sqrt{Tc_T\gamma_T} \qquad K(\sigma_0) = \sqrt{\frac{1}{\log(1+\sigma_0^{-2})}},
\end{aligned}$$

where the $(\star)$ follows from (Krause and Ong, 2011, Theorem 5) and an intuitive way to understand it is that the total information gain (i.e. the predictive variance term; see (Srinivas et al., 2009, Lemma 5.3)) is bounded by the maximum information gain under the optimal design.

It remains to bound γ_T for the kernel κ. Since κ is a tensor product of κ_1 and κ_2, with κ_2 being a kernel on a discrete set with cardinality 2^s (i.e. has rank 2^s), according to (Krause and Ong, 2011, Theorem 2),

$$\gamma_T \le 2^s(\gamma_T|_{\kappa_1} + \log T),$$

where $\gamma_T|_{\kappa_1}$ is the maximum information gain defined for the GP with kernel function κ_1. Note κ_1 is the Gaussian kernel. (Srinivas et al., 2009, Theorem 5) tells us that $\gamma_T|_{\kappa_1} = \mathcal{O}((\log T)^{d+1})$, where the implicit constant depends on the domain $\mathcal{X}$. Hence, $\gamma_T \lesssim 2^s(\log T)^{d+1}$. Plugging this into the above bound for R_T yields the high-probability bound (21). For (22), note that (21) and $\sqrt{x+y} \le \sqrt{x} + \sqrt{y}, x, y \ge 0$ together imply that there exists an absolute constant $C > 0$ so that with probability at least $1-\delta$,

$$\widetilde{R}_T := \frac{\left|R_T - CK(\sigma_0)\sqrt{2^sT(\log T)^{d+1}\log(2^sT^2)}\right|}{CK(\sigma_0)\sqrt{2^sT(\log T)^{d+1}}} \le \sqrt{\log\left(\frac{1}{\delta}\right)},$$

i.e., $\mathbf{P}(\widetilde{R}_T \ge x|\{\boldsymbol{\beta}_t\}) \le e^{-x^2}$. Integrating the tail probability yields

$$\mathbb{E}[\widetilde{R}_T|\{\boldsymbol{\beta}_t\}] = \int_0^\infty \mathbf{P}(\widetilde{R}_T \ge x|\{\boldsymbol{\beta}_t\})dx \le \int_0^\infty e^{-x^2}dx = \sqrt{\pi}.$$

Taking expectation over $\{\boldsymbol{\beta}_t\}$ yields $\mathbb{E}[\widetilde{R}_T] \le \sqrt{\pi}$. (22) follows by rearrangement.

For a_t chosen according to the Thompson sampling, we employ a similar technique that appears in (Lattimore and Szepesvári, 2020, Theorem 36.1). First, note that for any two random variables Z_t, Z'_t with the same mean,

$$\mathbb{E}[f(\boldsymbol{\beta}_t, a_t^*) - f(\boldsymbol{\beta}_t, a_t)] = \mathbb{E}[f(\boldsymbol{\beta}_t, a_t^*) - Z_t + Z'_t - f(\boldsymbol{\beta}_t, a_t)].$$

Conditioning on the historical actions $\{a_s\}_{1\le s\le t-1}$ and rewards $\{r_s\}_{1\le s\le t-1}$ up to $t-1$ (i.e. the σ-field $\mathcal{F}_{t-1}$), a_t and a_t^* are the argmax of $f(\boldsymbol{\beta}_t, a)$ and $f'(\boldsymbol{\beta}_t, a)$, respectively, where $f(\boldsymbol{\beta}_t, a)$ and $f'(\boldsymbol{\beta}_t, a)$ have the same distribution (i.e. posterior distribution of f). As a result, a_t and a_t^* have the same $\mathcal{F}_{t-1}$-conditional distribution. Now take Z_t and Z'_t as the UCB scores of a_t^* and a_t at $t-1$, respectively:

$$Z_t = \mu_{t-1}(\boldsymbol{\beta}_t, a_t^*) + c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t^*) \qquad\qquad Z'_t = \mu_{t-1}(\boldsymbol{\beta}_t, a_t) + c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a_t).$$

It is easy to verify using the tower property that $\mathbb{E}[Z_t] = \mathbb{E}[\mathbb{E}[Z_t|\mathcal{F}_{t-1}]] = \mathbb{E}[\mathbb{E}[Z'_t|\mathcal{F}_{t-1}]] = \mathbb{E}[Z_{t-1}]$. On the other hand, according to (23), it holds with probability at least $1-2\delta$ that

$$f(\boldsymbol{\beta}_t, a_t^*) - Z_t + Z'_t - f(\boldsymbol{\beta}_t, a_t) \le c_t^{1/2}\sigma_{t-1}(\boldsymbol{\beta}_t, a) \qquad\qquad t \in \mathbb{N}.$$

Using a similar analysis in the UCB case, we conclude that

$$\mathbb{E}[R_T] = \sum_{t=1}^{T}\mathbb{E}[f(\boldsymbol{\beta}_t, a_t^*) - Z_t + Z'_t - f(\boldsymbol{\beta}_t, a_t)] \lesssim \sqrt{\frac{2^s T(\log T)^{d+1}\log(2^s T^2)}{\log(1+\sigma_0^{-2})}}.$$

□

C Preliminary Study of the Interface Conditions

We conducted a preliminary study on a 2D Poisson equation $u_{xx} + u_{yy} = 1$ with the solution shown in Figure 9.

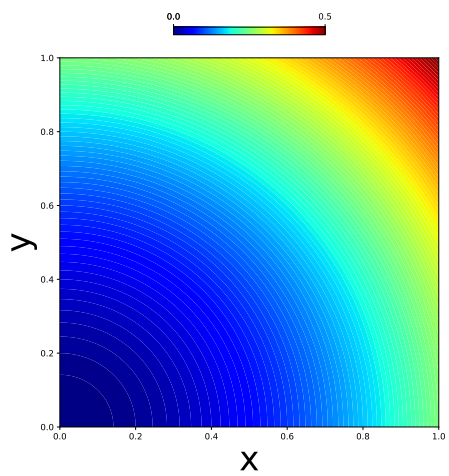
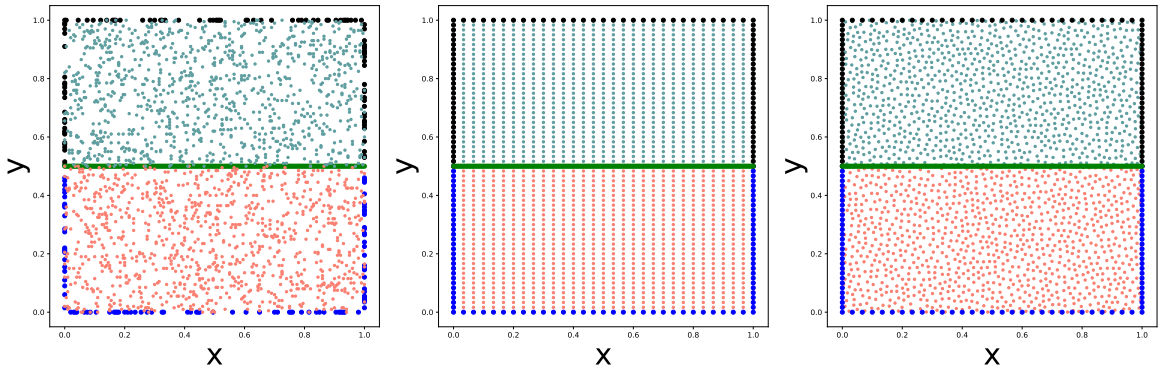


Figure 9: Poisson solution.

Given this PDE problem, we compared three types of boundary and collocation point sampling methods: random, grid, and Poisson disc sampling, as shown in Figure 10. The comparison was done between a standard PINN and XPINN, where the number of collocation points in each XPINN subdomain is the same as the total number of collocation points used by the PINN. We trained the two models with the boundary loss term weight λ_b set to 1 and 20. We also varied the interface loss term weight λ_I from $\{1, 20\}$. The interface loss term is computed from (13) and (15). Table 4 shows the L_2 relative error averaged over 10 runs to minimize the variance in network initialization and optimization. We can see that the XPINN performance is relevantly less variant to differences in sampling and weights, but for PINNs these differences result in order of magnitude changes in error. For this reason, we have conducted all the evaluations fairly by using random sampling and

larger boundary weights for PINN's which was the best overall setting. We also make the insight that random sampling allows a PINN to see higher frequencies according to the Nyquist-Shannon sampling theorem which may be the reason for increased performance over the other sampling methods. The XPINN includes an additional complexity of subdomains and interface conditions which may dominate the training, resulting in less variance as a function of collocation points.



(a) Random (b) Grid (c) Poisson Disc

Figure 10: Random, grid, and Poisson disc sampling for the Poisson equation problem. The interface is the green line at $y = 0.5$. Blue and black dots show the sampled boundary points in each subdomain, and the internal dots (red and cyan) the sampled points inside each subdomain.

Model	Random		Grid		Poisson Disc	
	$\lambda_{b,I} = 1$	$\lambda_{b,I} = 20$	$\lambda_{b,I} = 1$	$\lambda_{b,I} = 20$	$\lambda_{b,I} = 1$	$\lambda_{b,I} = 20$
PINN	9.0325e-4	**4.3223e-4**	6.0278e-3	2.3699e-3	5.3902e-3	2.2375e-3
XPINN	5.0884e-3	5.3205e-3	6.0764e-3	4.7829e-3	6.5061e-3	4.4888e-3

Table 4: Average L_2 relative error over 10 runs for different sampling techniques and loss term weights.

C.1 Interface Condition Combination

For the same PDE problem with random sampling, we ran multi-domain PINNs with different sets of conditions. We used the generalized interface condition notations for multi-domain PINNs as described in Table 3. For example, an XPINN can be described as $I_{u_{avg}} + I_{rc}$. The weights on all terms are unity. As seen by the results in Table 5, the multi-domain PINNs with interfaces $I_{u_{avg}} + I_c + I_{yy}$ outperforms other combinations as well as the PINN. We can see that with the correct interface conditions, the multi-domain PINN can greatly improve upon the standard XPINN. In fact, the additional residual continuity term, a trait of XPINNs, performs infinitesimally better than only using the average solution continuity. These results are the foundation of the METALIC method as we have shown that different combinations of conditions result in drastically different performances. We can also see that multi-domain PINNs are more general and flexible than the existing PINN decomposition models such as XPINN and cPINN. Having only used XPINNs in Table 4, one might conclude decomposing this problem is inferior to a standard PINN. However, we have shown that cPINN outperformed XPINN by an order of magnitude and that adding the additional term I_{yy} improved the cPINN even further. Furthermore, naively adding all possible terms such as in the final row, does not necessarily give the best results. This leaves three options for multi-domain PINNs: manually tuning the interface conditions, running all possible permutations such as we have done here, or devise a method to learn the appropriate interfaces such as METALIC.

Model	L_2 Relative Error
PINN	1.05e-3 ± 4.38e-4
$I_{u_{avg}}$	4.28e-3 ± 2.63e-3
$I_{u_{avg}} + I_{rc}$	3.92e-3 ± 2.25e-3
$I_{u_{avg}} + I_c$	9.45e-4 ± 2.85e-4
$I_{u_{avg}} + I_{rc} + I_c$	9.77e-4 ± 3.45e-4
$I_{u_{avg}} + I_{rc} + I_{gr}$	4.57e-3 ± 3.18e-3
$I_{u_{avg}} + I_c + I_{yy}$	**5.26e-4 ± 1.97e-4**
$I_{u_{avg}} + I_{rc} + I_{gr} + I_c + I_{yy}$	9.34e-4 ± 3.18e-4

Table 5: Average L_2 relative error over 10 runs for different interface combinations. Note: $I_c = I_y$ for this problem.

D Meta Learning Result Analysis

There are $2^9 = 512$ possible combinations of the interface conditions. For convenience, we use an integer to index each configuration (combination), index $= \sum_{i=0}^{n} 2^i \mathbf{c}[i]$ where $\mathbf{c}$ is a list of binaries, and $\mathbf{c}[i] = 1$ means i-th interface condition is turned on. We therefore can show how different sets of interface conditions are selected along with the equation parameters (see Figures 11, 14, 17, and 20).

D.1 Poisson Equation

For each PDE test case, we provide three analysis plots to better understand the METALIC results. For the Poisson problem, Figure 11 provides an overview of the interface configuration groupings as the equation parameter s varies. As opposed to Random-Single, the various METALIC methods predict interface configurations in groupings based on parameter s. This indicates that for these ranges, the PDE solution behaves similarly across the interfaces. It can also be seen that the configurations chosen between METALIC-Single and METALIC-Seq are different, indicating that the optimization is an important factor. This is logical since at the beginning of training, the PINN must first propagate information from the initial and boundary conditions inward to the entire domain. Therefore, interface conditions during this phase may in fact make learning more difficult in terms of the loss landscape as the network is trying to enforce continuity at a location which has no information but is simply a set of random predictions given the random initialization of weights and bias of the network.

In Figure 12, we can see the number of times the interface conditions are selected over the 100 test cases. Random-Single serves as a baseline with each interface being chosen roughly half of the time. The two most noticeable trends are that the gradient-enhanced residual term is almost never chosen and the flux continuity which is equivalent to u_y for this case is always chosen by METALIC. This is interesting as the gradient term in the original gPINN paper was shown to be beneficial to PINN training but appears to be a poor choice on a set of interface points, possibly because with all the other terms, it simply make the loss landscape more complex and does not provide a significant accuracy benefit compared to the other more theoretically sound terms such as flux. This result is novel as it showcases the robustness of METALIC in being able to distinguish between valid and invalid terms, something that would take a user doing manual tuning of these terms much trail and error to determine. We also note that the METALIC choices align with our results in Section C.1 that the flux conditions from cPINN greatly outperforms the residual continuity conditions in XPINN for Poisson's equation. Flux continuity is a well studied conservation term rooted in traditional methods whereas residual continuity is a term devised with the convenience of PINNs and automatic-differentiation(AD) in mind. We also note that when comparing the METALIC-Seq-UCB ADAM and L-BFGS choices, L-BFGS uses more terms on average than ADAM. This confirms our hypothesis from Figure 11 that more interface terms at the start of training may in-fact make training more difficult. This validates the result that not only is a sequential interface predictor more accurate, but also faster as it adds in terms when needed which would reduce computational cost. We also not that including the residual points in the overall set of collocation points is rarely chosen, likely due to the fact that the interface point set is an order of magnitude smaller than the collocation point set so assuming it is well sampled, its contribution is negligible. All these insights further confirm the method is working well and is consistent with our intuition and the properties of the equation being solved.

Finally, in Figure 13, we show the L^2 relative error as a function of s. This is a more detailed version of the error table in the manuscript which tells us how the problem difficulty changes over the parameterization of the problem. For this Poisson

problem, it is quite consistent other than the lower bound around $s = 1$ in which the forcing term is very smooth and as we expect the problem is quite simple, as reflected by the lower errors there.

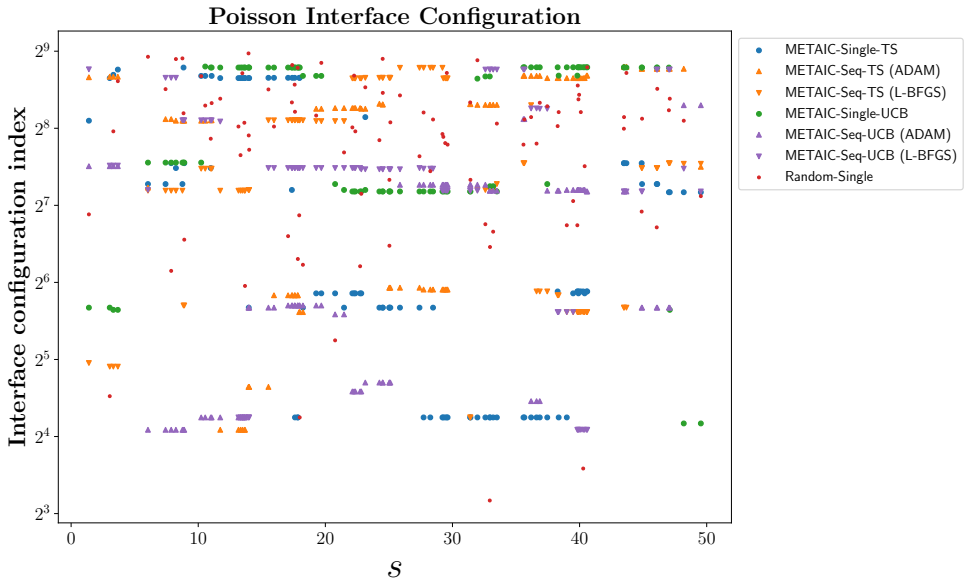


Figure 11: Scatter plot of interface configuration *vs.* the equation parameter s.

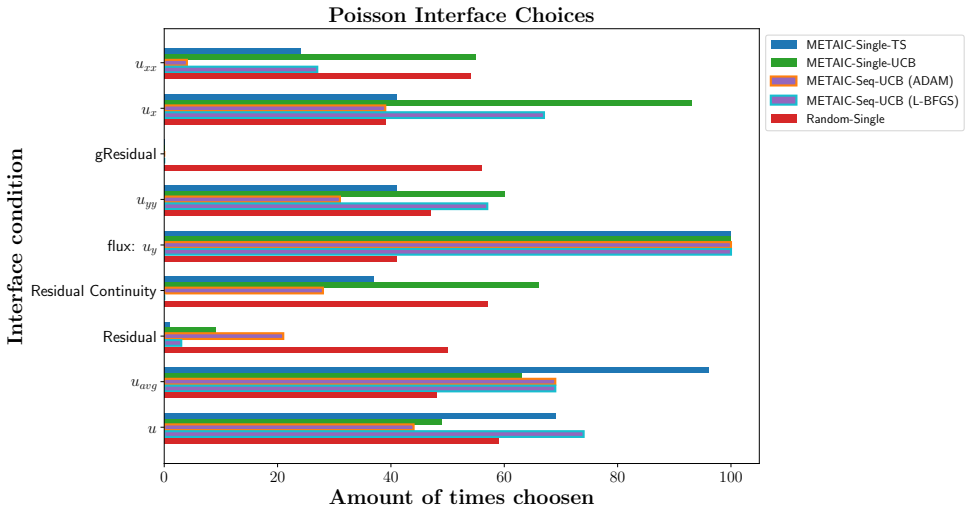


Figure 12: Horizontal bar plot of the quantity of interfaces chosen throughout testing over 100 randomly drawn equation parameters.

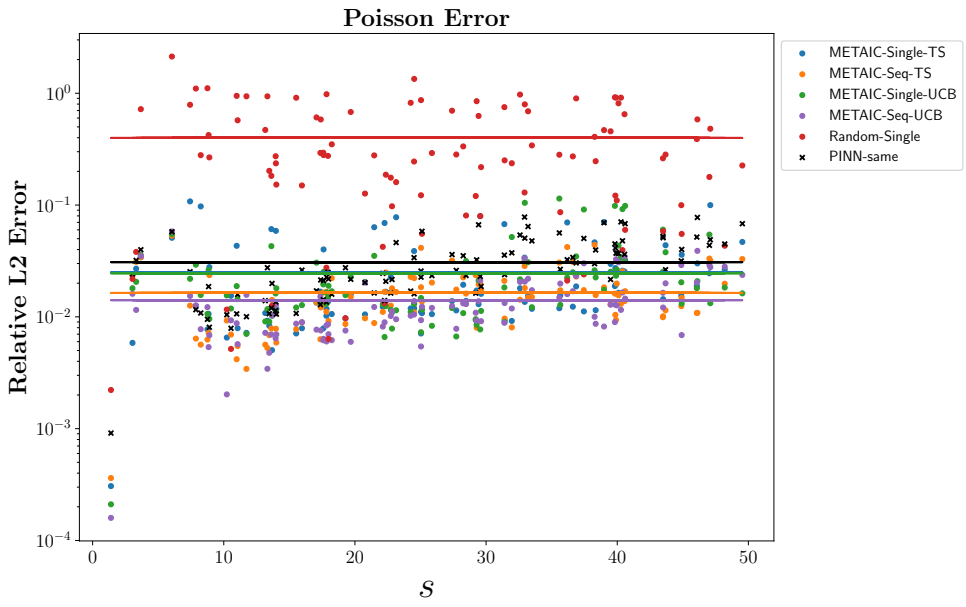


Figure 13: Scatter plot of the relative L_2 error *vs.* the equation parameter s.

D.2 Advection Equation

For the Advection problem, Figure 15 indicates that very few terms where needed. So much so that the ADAM step of METALIC-Seq-UCB has only one term, $I_{u_{avg}}$, the weaker form of the solution continuity. We again point out the benefit of METALIC in being able to sub-select few terms out of many while still resulting in the best accuracy as seen in Figures 14 & 16 which show that METALIC-Seq-UCB uses the fewest number of terms but has the best error. This emphasizes that more interface terms are not always better since the loss landscape can become more complex from an optimization standpoint. Another interesting feature is that the first derivative in space (u_x) is chosen more than the first derivative in time (u_t) despite the subdomain split being in time. This is opposite of the Poisson results in which the derivative normal to the interface (u_y), representing the flux, was chosen in all cases. Both terms, u_t and u_x are part of the PDE with flux simply being u, but the tangential derivative u_x appears to be a much more meaningful term when it comes to propagating the wave through the interface. There are also no second order terms which was the case with Poisson.

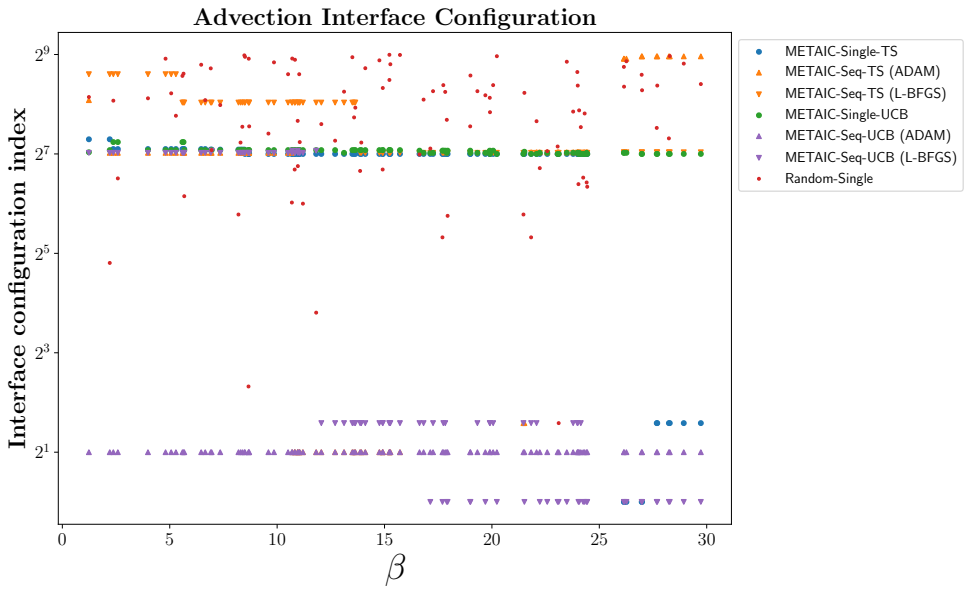


Figure 14: Scatter plot of interface configuration *vs.* the equation parameter β.

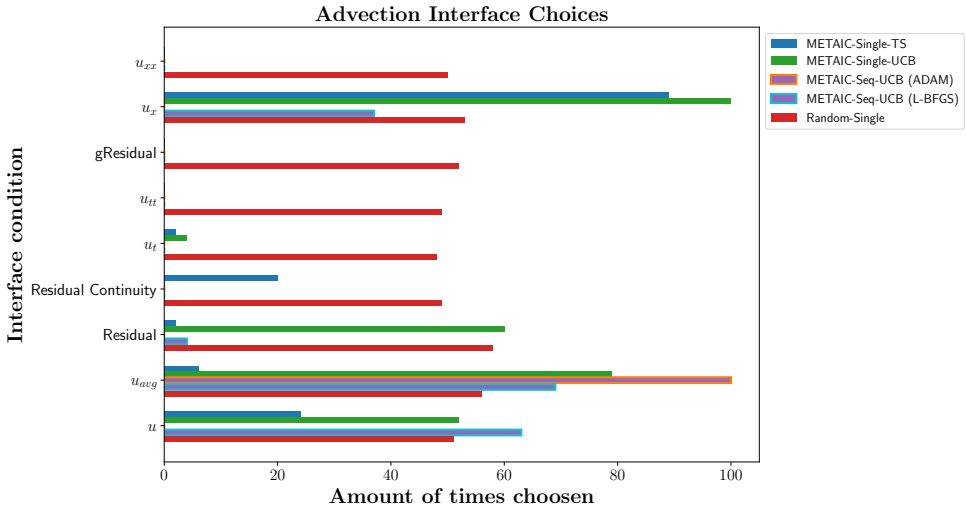


Figure 15: Horizontal bar plot of the quantity of interfaces chosen throughout testing over 100 randomly drawn equation parameters.

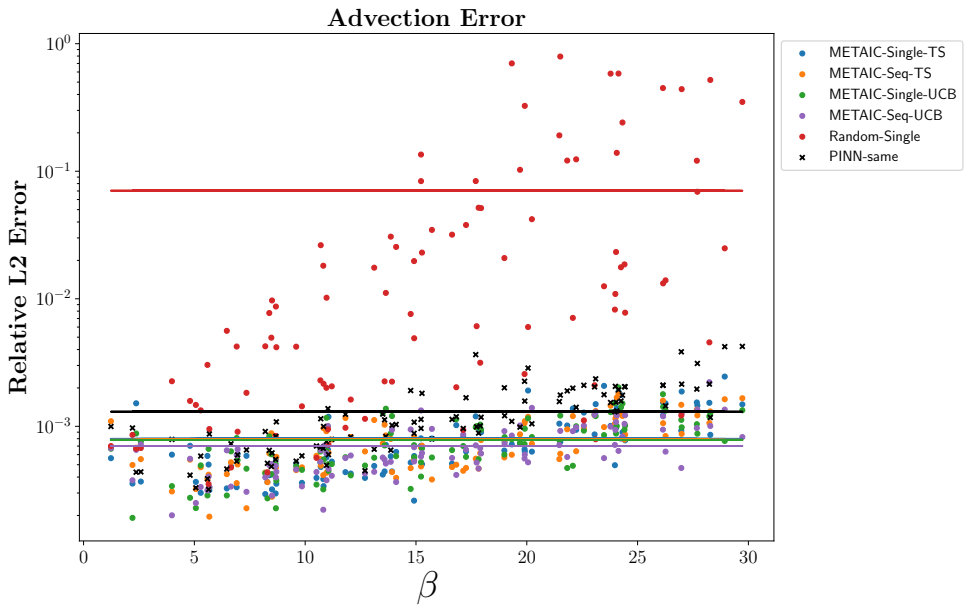


Figure 16: Scatter plot the relative L_2 error *vs.* the equation parameter β.

D.3 Reaction Equation

For the Reaction problem, we note that while this is an ODE, with no spatial derivatives, they were counter-intuitively chosen as interfaces. This emphasizes the fact that PINNs, and machine learning techniques in general, do not work the same as traditional methods since these terms are not necessary for well-posedness of an ODE. Given this, Figure 19 shows that METALIC outperformed the PINN while using these conditions. This is an interesting line of investigation for future work as it shows counter-intuitive terms can provide a training benefit to PINNs even in contrast to the previous Advection problem where almost no terms where chosen. It is not clear why in some cases only the most basic of terms are used while in others terms which do not make physical sense are chosen, but in both, the accuracy is quite good on their respective problems. This shows that METALIC learns something about PINN training that is not evidently clear to the human user.

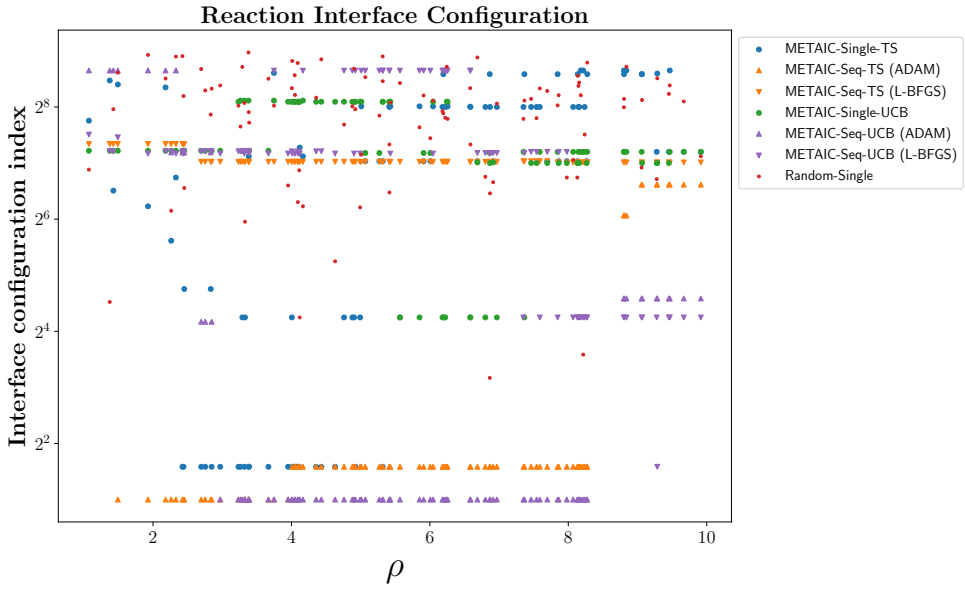


Figure 17: Scatter plot of interface configuration *vs.* the equation parameter ρ.

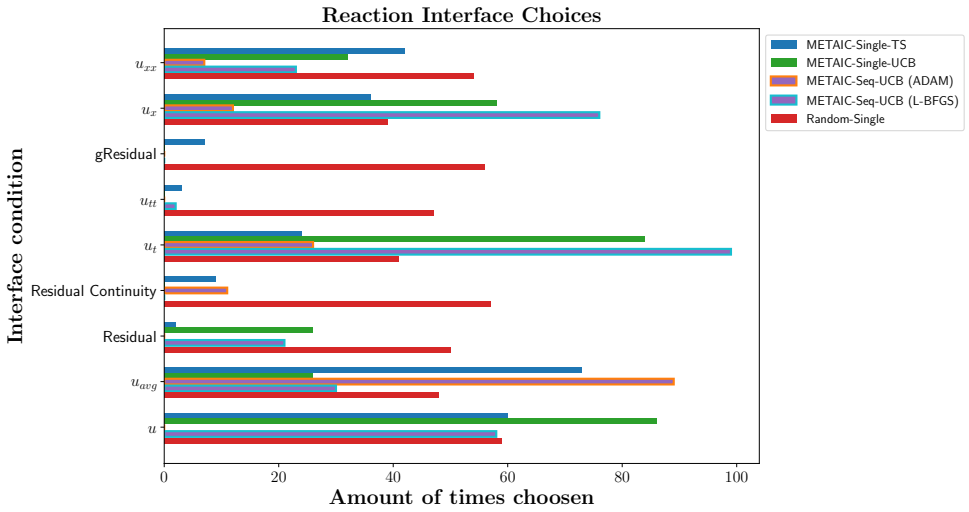


Figure 18: Horizontal bar plot of the quantity of interfaces chosen throughout testing over 100 randomly drawn equation parameters.

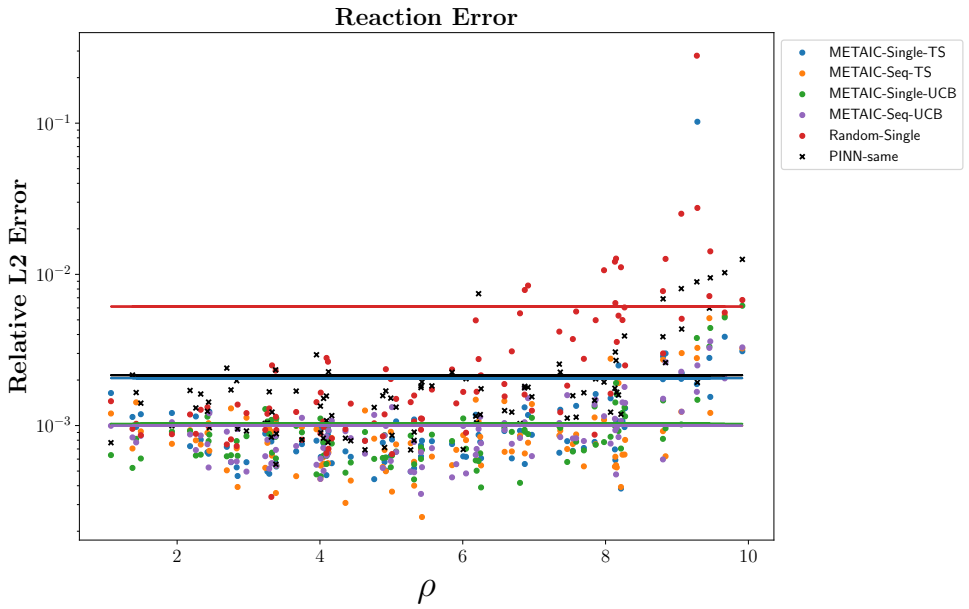


Figure 19: Scatter plot of the relative L_2 error *vs.* the equation parameter ρ.

D.4 Burger's Equation

For the Burger's problem, we see more of what one might expect from traditional interface continuity terms. Figure 21 shows that the flux term is predominately chosen, just as in Poisson's equation. Although here we see the flux is not equivalent to the first order derivative, enforcing the idea that it is in fact the flux providing the training benefit and not a coincidence of the first-order derivative and flux being the same for Poisson. We also see the largest improvement in error of METALIC over PINNs as seen in Figure 22. The trend is also consistent with our physical understanding, as viscosity (ν) increases the problem becomes more simple. This is because at lower viscosities a shock forms and creates a discontinuity in the solution which is difficult for PINNs to resolve. The decomposition of this problem is therefore the most sound, in that we allow one network to handle the sharp discontinuity in the center, and another to handle the relatively simple solution around it. This allows for the network in the center to learn a higher frequency basis with which to approximate the discontinuity instead of

also having to fit the lower frequencies around it which has been delegated to the second network. Given this, it makes sense that a multi-domain PINN with the METALIC method greatly outperform PINNs here. It also emphasizes that in the less physically motivated decompositions for Poisson, Advection, and Reaction, we still see improvement using multi-domain PINNs and METALIC.

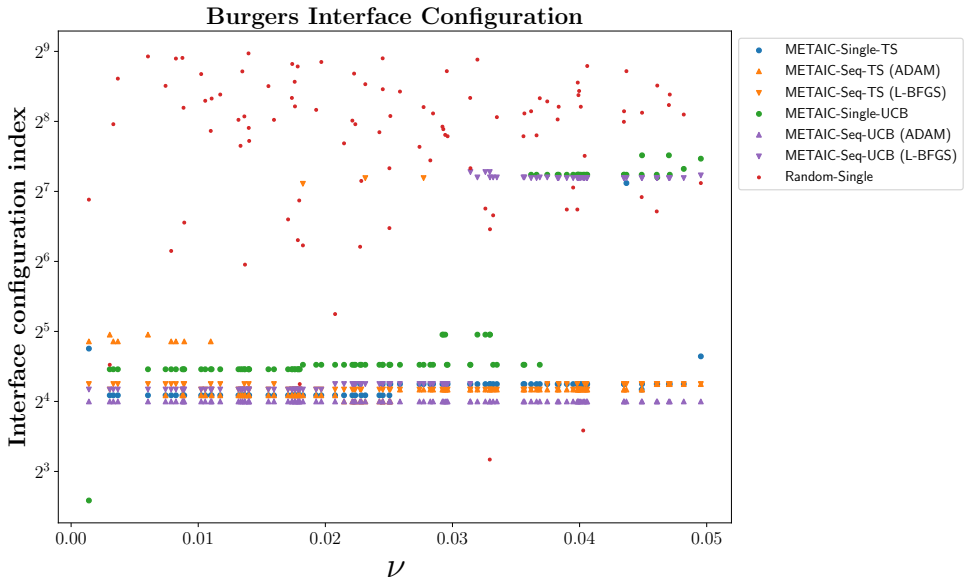


Figure 20: Scatter plot of interface configuration *vs.* the equation parameter ν.

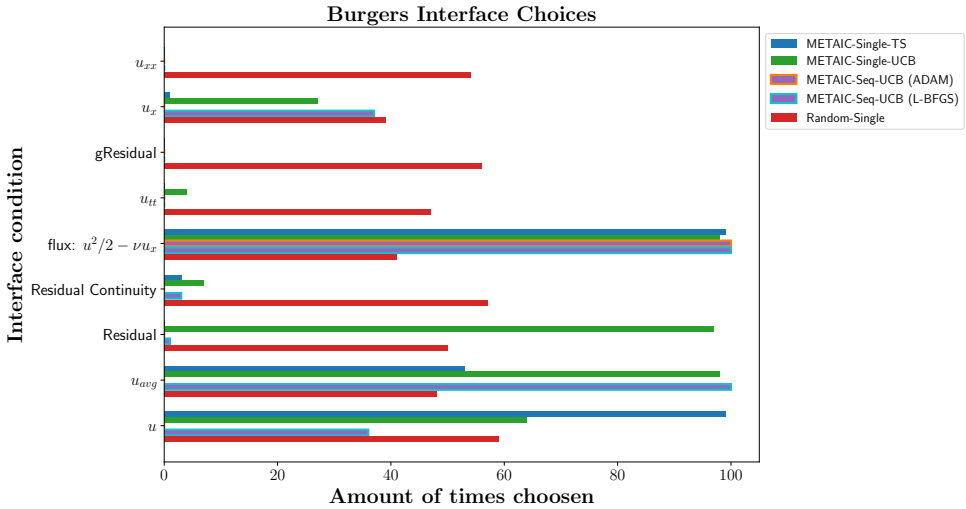


Figure 21: Horizontal bar plot of the quantity of interfaces chosen throughout testing over 100 randomly drawn equation parameters.

Figure 22: Scatter plot of the relative L_2 error *vs.* the equation parameter ν.