PIHLoRA: Physics-informed hypernetworks for low-ranked adaptation

Ritam Majumdar
TCS Research
ritam.majumdar@tcs.com

Vishal Jadhav
TCS Research
vi.suja@tcs.com

Anirudh Deodhar
TCS Research
anirudh.deodhar@tcs.com

Shirish Karande
TCS Research
shirish.karande@tcs.com

Lovekesh Vig
TCS Research
lovekesh.vig@tcs.com

Venkataramana Runkana
TCS Research
venkat.runkana@tcs.com

Abstract

Physics-informed neural networks (PINNs) have been widely used to develop neural surrogates for solutions of Partial Differential Equations. A drawback of PINNs is that they have to be retrained with every change in initial-boundary conditions and PDE coefficients. The Hypernetwork, a model-based meta learning technique, takes in a parameterized task embedding as input and predicts the weights of PINN as output. Predicting weights of a neural network however, is a high-dimensional regression problem, and thus it is observed that hypernetworks perform sub-optimally while predicting parameters for large base networks. In this work we investigate whether we can circumvent the above issue with use of low ranked adaptation (LORA). Specifically, we use low ranked adaptation to decompose every layer of the base network into low-ranked tensors and use hypernetworks to predict the low-ranked tensors. However, we observe that the reduced dimensionality of the resulting weight-regression problem does not suffice to train the hypernetwork well. Nevertheless, addition of physics informed loss (HyperPINN) drastically improves the generalization capabilities. In order to show the efficacy of our proposed methods we consider widely used PDEs used in the domain of Material Science such as Maxwell's equation, Elasticity equation, Burger's equation, Navier-Stokes. We observe that LoRA-based HyperPINN (PIHLoRA) training allows us to learn fast solutions while having an 8x reduction in prediction parameters on average without compromising on accuracy when compared to all other baselines.

1 Introduction

In the field of material science, Partial Differential Equations (PDEs) play a crucial role in modeling and understanding the properties and behaviors of materials. The Helmholtz equation [1–5] is extensively used in the design of metamaterials, solving problems related to electromagnetic properties and graded media materials. Structural mechanics heavily rely on PDEs like the Biharmonic equations [6] of elasticity and Euler-Bernoulli equations [7, 8] to simulate the behavior of materials under various stress conditions. Additionally, the diffusion equation [9, 10] is fundamental in describing substance distribution in different states of matter [11, 12], and Burgers' equation is pivotal in modeling shock wave interactions in nonlinear acoustics and high-viscosity material flows [13, 14]. These are examples of equations offer deep insights into complex material behaviors and are instrumental in advancing material science.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Most systems of PDEs are not amenable to analytical solutions and require numerical approximations. In recent years there has been a strong interest in using machine learning techniques for obtaining numerical approximations. One of the popular approaches of interest has been physics informed neural networks (PINNs) [15], where the solution for a PDE is approximated with a neural surrogate, also refered in this work as an implicit neural respresentation (INR). Representation of the final solution with a neural network can pottentially allow one to overcome several limitations that occur in numerical methods on account of discretization. We refer interesterd readers to [16, 17] for additional discussion.

PINNs despite their popularity suffer from several limitations, one of them being that they have to be retrained from scratch for every change in intial condition, boundary conditions or PDE coeffecients. A possible solution to overcome this challenge is to employ hypernetworks. Hypernetwork [18] is a form model based meta learning network that takes task descriptions as input and predicts the weights of a neural network as output. Predicting weights of a neural network however, is a high-dimensional regression problem, and hypernetworks perform sub-optimally while predicting parameters for large base networks. However, one can represent the adaptation of large network with parameter efficient tuning [19, 20]. Thus, a hypernetwork can be trained using just adaptation parameters rather than the entire network. Almost concurrent to our work such ideas have been used for Image Generation [21] and Instruction Tuning of LLMs [22].

In this work we show the efficacy of utilizing Hypernetworks to predict low rank adaptation of the implicit nerual represtations of the solutions for PDEs. We observe that hypernetworks trained purely by using Implicit Neural Representations (INRs) as data do not provide the best performance and significant performance gains can be obtained if one can finetune the weights of the INRs while training the hypernetworks as well. We believe that this performance difference may be observed due to permutation equivariance related issues [23]. There are two possible approaches to finetune the INRs while training the hypernetworks as well: (1) If one has access to input-output data for the neural surrogates, i.e. INRs, they can be used to define supervision loss. Such data may be available from sensors or from employing a PINN solver for each task. (2) HyperPINNs: One can introduce a physics informed loss instead or in addition to the above supervision loss. We observed the best performance when a physics informed loss is included during hypernetwork training.

The remainder of the paper is organized as follows. In section 2, we describe the related work and some PDEs relevant to the material science community. In section 3, we describe the methodology and state the motivations behind each technique we adopt. Sections 4 consists of the experiment design. We discuss the results and observations in section 5 and conclude in section 6.

2 Related Work

2.1 Parameter Efficient Finetuning

Parameter-efficient fine-tuning (PEFT) adapts pre-trained large language models (LLMs) by training a select subset of parameters. PEFT, encompassing various techniques such as adapter modules, prompt tuning, and sparse updates, is explored in [24] for its robust application across language tasks, and in [25] for enabling efficient task adaptation. [26] provides an extensive comparison of PEFT methods like prefix tuning and adapter layers, directing readers to this review for a detailed overview. Additionally, [27] highlights the use of transfer learning with adapters in NLP to overcome parameter inefficiency in traditional fine-tuning. Low-ranked adaptation, popularly known as LoRA, is one of the most popular techniques of PEFT. [19] addresses the challenges of adapting large-scale models like GPT-3 175B for specific tasks, where full fine-tuning is impractical due to the enormous number of parameters. Building upon this, [28] and [29] present systems that enhance speech recognition and reduce both trainable parameters and activation memory costs in LLMs. [30] proposes a quantization-aware adaptation algorithm for deploying LLMs in resource-limited environments, while [31] focuses on efficient adaptation through perturbation of weight matrices in a Bayesian context. Beyond NLP, [32] delves into the theoretical aspects and applications of LoRA in fine-tuning diffusion models. [33] introduces an approach for dynamic rank adaptation to overcome the rigidity of traditional LoRA, and [34] applies LoRA for solving visual problems with open-source LLMs, enhancing tool invocation accuracy and enabling zero-shot learning. Collectively, these studies demonstrate LoRA's wide-ranging efficacy and flexibility across various fields. Adapters is another form of PEFT. Adapters in neural networks provide a resource-efficient method for fine-tuning pre-trained

models across various computational fields, particularly in natural language processing (NLP). [35] enable parameter-efficient adjustments without full fine-tuning. "CHAPTER" [36] introduces CNN adapters for self-supervised learning in speech models, significantly reducing the parameters needed for fine-tuning. "AdapterHub" [37] proposes a framework for adapting transformer models in NLP, addressing storage and sharing challenges of large models. Similarly, [27] suggests adapters for transfer learning, enhancing model compactness and extensibility. "AdapterGNN" [38] applies these principles to GNNs, showing the adaptability of this approach beyond traditional NLP applications. Collectively, these studies underscore the efficiency and versatility of adapters in fine-tuning large, pre-trained models in a variety of domains.

2.2 Meta-learning for Partial Differential Equations

Meta-learning techniques have been applied to solve a range of PDEs with diverse equations, initial conditions, and boundary conditions, treating each variation as a distinct task. [39] introduced a shared, offline-learned loss function for different tasks within PINNs. [40] suggested using a reptile-initialization method for various PINN tasks. Meanwhile, [41] conducted a comprehensive analysis of various meta-learning initializations, showcasing their effectiveness across PDEs like Burgers', Allen-Cahn, and Diffusion-Reaction equations, particularly when dealing with changing equation coefficients. Hypernetworks [18] is a popular model-based Meta learning technique, have been briefly explored for solving PDEs. Hypernetworks use one network to generate weights for another, showing versatility across various applications [42]. Hypernetworks have been applied to graphics for to BRDF estimation [43], Neural Architecture Search [11], and HyperPINNs [44] to solve parameterized PDEs. In Physics Informed Symbolic Networks (PISN) [45], the authors employ hypernetworks to predict differentiable programs and generate symbolic solutions for PDE. Moreover, INRs have become vital in encoding signals for diverse applications like medical imaging, scene reconstruction [46], and efficiently solving parametrized PDEs [47]. In HyperDreamBooth [21], the authors combine hypernetworks with LoRA in diffusion models, demonstrating their potential in personalized applications.

2.3 Partial Differential Equations for Material Science

Helmholtz Maxwell Equation: The study and design of metamaterials, which are engineered to have properties not found in naturally occurring materials, often involve solving the Helmholtz equation. The authors in [1] apply a PINN to predict the effective material parameters of a two-dimensional metamaterial from its scattered fields by employing a loss function based on the Helmholtz equation. [2] uses a PINN to design an electromagnetic metamaterial based on high-frequency Helmholtz equation. [3–5] consist of additional examples of using Helmholtz equation to model graded media materials.

Structural Mechanics: PDEs are used to model the behavior of materials under various loads and conditions. [6] develops a surrogate model to solve Biharmonic equations of elasticity using PINNs. Elasticity equations are based on Hooke's law to develop stress-strain relations. [7] develops PINNs to simulate complex structural systems consisting of single and double beams governed by Euler-Bernoulli and Timoshenko PDE. [8] uses PINNs to model the strain-rate and temperature dependence of the deformation fields in elastic-viscoplastic solids.

Diffusion Equation: In material science, the diffusion equation is a fundamental PDE that describes the distribution of a substance within a system over time, essential for understanding phenomena like heat conduction, mass transfer and the diffusion of particles in solids, liquids, or gases. [48] solves anisotropic diffusion equation using PINNs.[12] models multi-component Diffusion reaction equation using PINNs. [9, 10] develops neural surrogate solutions for non-Fickian coupled diffusion elastic wave equation.

Burger's equation: Burgers' equation is used to model shock waves and their interactions in various materials. It helps understand how these waves evolve and dissipate over time and distance, particularly in nonlinear acoustics. In highly viscous materials like polymers [14] or glass [13] during manufacturing, Burgers' equation can model flow behavior, especially when there are nonlinear effects due to the material's viscosity.

3 Methodology

3.1 Physics-informed Neural Networks

We consider Partial Differential Equations of the form

$$N[x, t, u(x, t)] = 0, t \in [0, T], x \in \Omega \tag{1}$$

Here, N is a non-linear differential operator consisting of space and time derivatives of spatial variable x and temporal variable t. The neural network is trained using a loss function that consists of: 1) A supervised initial condition loss, boundary condition loss and 2) An unsupervised physics-informed loss component.

$$L(\theta) = \sum_{x_i \in IC} (u(x_i, 0) - f(x_i))^2 + \sum_{x_{bc}, t_{bc} \in BC} (u(x_{bc}, t_{bc}) - h(x_{bc}, t_{bc}))^2 + \sum_{x,t \in C} N(u, x, t, u_t, u_x, ...)^2 \tag{2}$$

Here, C represents the set of collocation points on which the Physics-loss is computed. IC, BC represents the set of points at which Initial Conditions and Boundary Conditions are evaluated. f,h, represent the initial condition and boundary condition functions respectively. θ represents the weights of the neural network.

3.2 HyperPINNs: Overcoming the Limitations of Standard PINNs

While PINNs have proven effective as neural surrogates for solving instances of PDEs, they are typically tailored to a single PDE instance and lack generalizability across multiple instances. This limitation is where HyperPINNs come into play, combining the principles of hypernetworks with PINNs to address this challenge.

3.2.1 Hypernetworks

Hypernetworks [18], a model-based meta-learning technique used to handle multiple instances of tasks. In this framework, a hypernetwork H, parameterized by weights θ_h, accepts a task representation λ as input. This hypernetwork predicts the weights θ_m for a base neural network M, which is then used to model a specific instance of a task. The equation governing this relationship is:

$$\theta_m = H(\lambda; \theta_h) \qquad \mathbf{u}(x, t) = M(x, t; \theta_m) \tag{3}$$

In this setup, θ_m is supplied directly to the main network, facilitating the evaluation of the solution $\mathbf{u}$. This method enables the hypernetwork to be trained on various samples of training tasks λ, empowering it to generalize to unseen tasks during testing. The key advantage here is the elimination of the need to train neural networks from scratch for each new task instance.

3.2.2 HyperPINNs

Building upon this foundation, HyperPINNs, introduced in [44], are specialized hypernetworks trained with a physics-informed loss function, specifically designed to tackle parameterized PDEs:

$$\theta_m = H_{pinn}(\lambda; \theta_h) \qquad \mathbf{u}_{pinn}(x, t) = M_{pinn}(x, t; \theta_m) \tag{4}$$

The architecture of HyperPINNs consists of two core components:

Hypernetwork (H_{pinn}): This component takes the task parameterization λ_h as input and outputs the weights θ_m for the main network.

Main Network (M_{pinn}): This network utilizes the weights θ_m to solve specific instance of a PDE.

The introduction of HyperPINNs effectively mitigates the primary drawback of conventional PINNs—the necessity of retraining for each new PDE instance, by directly predicting the weights for the main network. While hypernetworks handle various instances of tasks, they encounter a significant challenge when scaling up: the complexity of the weight regression problem increases

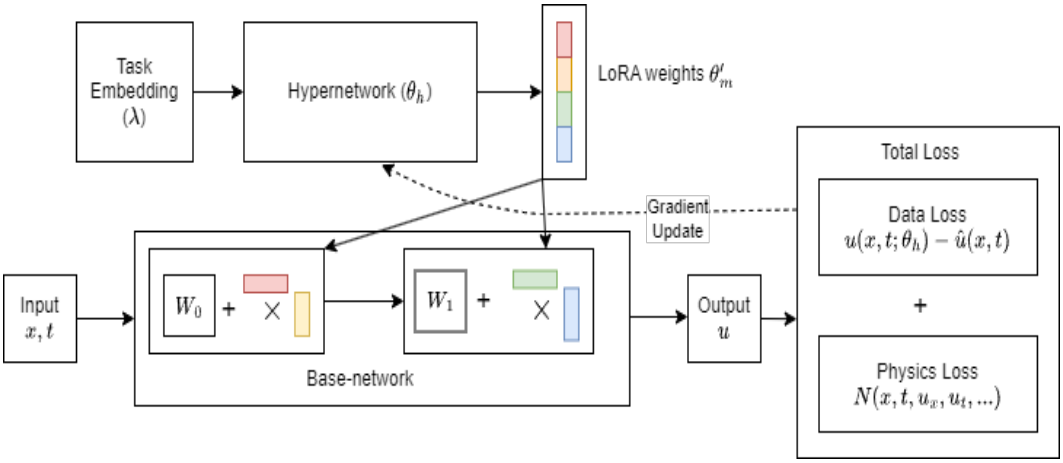

Figure 1: PIHLoRA. The hypernetwork parameterized by θ_h, takes in task embedding λ as input, and outputs Low-ranked weights θ'_m. They are passed into the base-network (We show 2-layers as reference) with pre-trained weights W_0, W_1. The base network takes in variables of PDE x, t as input and outputs u, which is trained in a physics-informed manner to update θ_h.

substantially with the size of the main network. This complexity arises from the high dimensionality of the weight space that the hypernetwork must navigate, making the process increasingly difficult for larger base networks. To address this issue, we use Low-Rank Adaptation (LoRA) and intergrate it with PINNs and HyperPINNs. LoRA decomposes the weight matrices of the main network, reducing the dimensionality of the weight regression problem. The mathematical formulations are described as follows:

3.3 Low ranked adaptation for PINNs (L-PINNs)

Building on the methodology proposed in [19], we begin by training a PINN on a baseline task T_0, denoting its weights as W_0. For a new task T_1, the network's weights W_1 are updated as follows:

$$W_1 = W_0 + \Delta W = W_0 + A \cdot B \tag{5}$$

In this formulation, $W_1 \in \mathbb{R}^{m \times n}$ and $W_0 \in \mathbb{R}^{m \times n}$, with $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, where r is less than $\min\{m, n\}$. During training, W_0 remains fixed, while matrices A and B are trainable, allowing adaptation from task T_0 to T_1 with reduced parameter complexity.

3.4 Bridging Hypernetworks and LoRA: HLoRA and PIHLoRA

3.4.1 HLoRA: LoRA-Based Hypernetworks

Extending this concept to hypernetworks, we introduce LoRA-based Hypernetworks (HLoRA). Here, the hypernetwork H, parameterized by θ_h, predicts low-rank weight adjustments A, B (denoted as $\theta'_m = [A : B]$) instead of the full weight matrix. The formulation is as follows:

$$\theta'_m = H(\lambda; \theta_h) \qquad \theta_m = W_0 + \theta'_m \qquad \mathbf{u}(x,t) = M(x,t;\theta_m) \tag{6}$$

3.4.2 PIHLoRA: LoRA-Based HyperPINNs

Taking this advancement further, we combine LoRA-based adaptation with HyperPINNs to form LoRA-based HyperPINNs (PIHLoRA), designed to solve parameterized PDEs efficiently. The mathematical formulation is:

$$\theta'_m = H_{pinn}(\lambda; \theta_h) \qquad \theta_m = W_0 + \theta'_m \qquad \mathbf{u}(x,t) = M_{pinn}(x,t;\theta_m) \tag{7}$$

We provide an illustration of LoRA-based HyperPINN in figure 1. This approach enables the efficient solving of parameterized PDEs by predicting fewer parameters compared to standard HyperPINNs, thereby addressing scalability and adaptability challenges inherent in high-dimensional weight spaces.

4 Experiment Design

Our experiment design is structured into two distinct sections. The first section focuses on assessing the impact of varying the rank in the decomposed weight matrices on the performance of PINNs. This exploration is critical, as it aims to identify the optimal balance of parameters necessary for our subsequent hypernetwork experiments. Essentially, decomposing the weights into higher ranks increases the network's capacity to represent and adapt to the dynamics of new tasks in relation to the original task. However, this also means that the hypernetwork must predict a larger number of parameters. Conversely, a lower rank decomposition simplifies the hypernetwork's prediction task by reducing the number of parameters but potentially compromises on the network's ability to accurately represent the dynamics of the original task. Two baselines for these experiments are PINNs trained from scratch and PINNs finetuned over a base task (F-PINNs), without decomposition.

In the second section of our experiments, we delve into hypernetwork-centric investigations, structured into three key benchmarks, each with a distinct focus:

True Solution-Based (TSB): The initial benchmark involves training hypernetworks using actual PDE analytical solutions as outputs. In specific cases like the 1D-Burger's equation, we employ ground-truth simulations from numerical solvers as the training labels. While this method relies on the availability of true simulations and analytical solutions, which are often scarce in practical applications, it establishes a foundational baseline for our experiments.

INR Weight Regression (IWR): Addressing the limitation of the first benchmark, the second benchmark takes a different route. Here, both F-PINNs and L-PINNs are trained on a large dataset offline, from which we collect the trained weights. A hypernetwork is then trained to regress on these Implicit Neural Representation (INR) labels, as inspired by [49]. This method, while computationally intensive due to the extensive offline PINN training, marks a significant improvement over the first benchmark by reducing dependency on analytical solutions.

PINN Output Inference (POI): In the third benchmark, we utilize the solutions inferred by the trained PINNs as ground truth for training the hypernetworks. Similar to the OPR benchmark, this approach is computationally demanding, especially given the volume of offline PINN training involved, but it further refines our methodology by integrating practical inference scenarios.

Finally, our primary experiment, Physics-Informed Hypernetworks with Low-Rank Adaptation (PIHLoRA), synthesizes the insights gained from the previous benchmarks. It directly combines Physics-Informed Hypernetworks with LoRA, eliminating the need for generating an extensive offline dataset. This experiment represents the culmination of our efforts, integrating and validating the observations from the rank analysis across all benchmarks.

5 Results and Observations

5.1 Rank analysis of decomposed Low-rank matrices

Figure 5.1 describes the rank analysis results for LoRA experiments, and a detailed tabulation of the results in Table 2. Across all PDE examples, we first train an independent PINN on a task T_0. Then we perform Low-ranked adaptation for a new task T_x using the trained PINN for task T_0 as our base model. Lower the rank of the decomposed tensors, the fewer the number of parameters involved in adapting to the newer task.

Across all PDE systems, we notice the average accuracy across the test-tasks first increases with increase in rank, reaches an optimal rank, and then slightly deteriorates as the rank is further increased to the full rank of the original matrix. We increase the rank of the matrices in multiples of 2. The best L-PINN is on average $4.58\times$ accurate than PINNs, while requiring $8.22\times$ fewer parameters, $4.79\times$ fewer epochs while yielding $5.25\times$ faster training time per epoch (due to training fewer parameters). Additionally, the best LoRA-based PINN is on average $2.92\times$ accurate than a finetuned-PINN, while requiring $8.22\times$ fewer parameters, $1.86\times$ fewer epochs yielding $5.25\times$ faster training time per epoch.

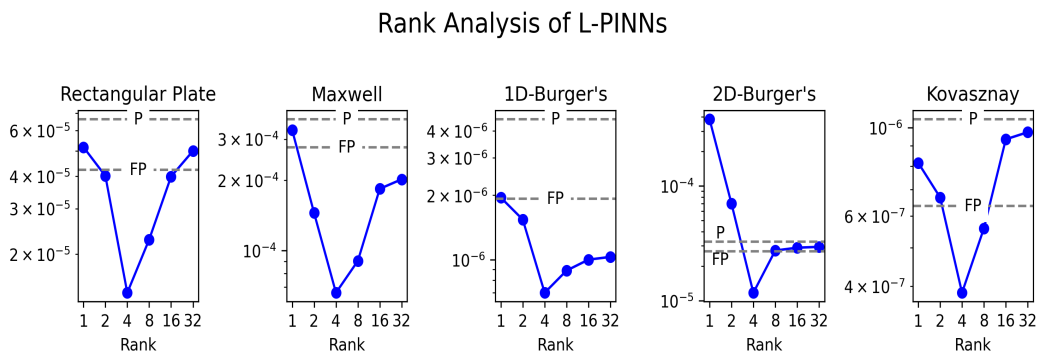

Figure 2: Rank analysis of PINNs. Here, the blue trendline indicates relationship between MSE and the rank of decomposed matrix. P and FP refers to the benchmarks, PINNs and Finetuned PINNs.

Improvement of performance due to initial rank increase can be attributed to improved representation capacity of the finetuned task T_x due to higher number of parameters. We see a 3.65%, 4.96%, 2.78%, 2.11%, 32.56% improvement in Rectangular Plate vibration, Maxwell, 1D-Burger's, Kovasznay, 2D-Burger's respectively on increasing the rank from 1 to 4. We additionally notice, while the training-time per epoch is lower for L-PINNs with ranks 1 and 2, as the representation capacity is restricted, these take a higher time to converge as compared to L-PINNs with rank 4. Further increase in rank leads to drop in performance, and we speculate overfitting and increase in variance to be reasons for it due to increase in # parameters. In addition, very high ranked L-PINNs require higher training time to converge, as against the best-ranked (4) L-PINN.

5.2 Hypernetworks for Low-ranked adaptation

Figure 5.1 describes the results of our hypernetwork experiments, and a detailed tabulation of the results in Tables 3, 4, 5, 6, 7. Across all examples, * refers to the case where we don't perform LoRA decomposition and hypernetwork predicts the weights of entire base network.

In our first benchmark, True Solution-Based (TSB), we trained HLoRA using traditional solvers for 1D-Burger's and analytical solutions for the remaining examples. Consistent with our previous analyses, hypernetwork performance improves up to a LoRA of rank 4, beyond which we observe a decline. This drop in performance for ranks higher than 4 is attributed to the hypernetworks' challenge in predicting high-dimensional weights in a regression setting. When tested, Rank-4 HLoRA show varying degrees of performance: being slightly inferior (2.69×,1.25x,1.35× worse) to L-PINNs in Rectangular plate, Maxwell, Kovasznay flow, respectively, but substantially less effective (10.15× and 460× worse) in 2D-Burger's and 1D-Burger's, respectively. Most practical PDE systems don't have a closed-form analytical solutions, and generating simulations using traditional PDE solvers can be computationally intensive.

To address these limitations, our second benchmark, INR Weight Regression (IWR), employs multiple instances of trained PINNs, using their weights as supervised labels for the hypernetwork. Here, the hypernetwork predicts the weights of the base network, a strategy inspired by [49]. Despite a similar rank analysis and reduced training time, the performance on test examples is notably poor. This is likely due to the extreme sensitivity of weight-spaces to minor perturbations, resulting in complex manifolds that are difficult to generalize. INR-regressor hypernetworks perform significantly worse (2-3 orders) compared to L-PINNs and the TSB benchmark.

In our third benchmark, PINN Output Inference (POI), we train multiple PINN instances, using them to generate ground truth labels over a computational domain. The hypernetwork then maps task embeddings to a neural network's weights, replicating the outputs produced by the trained PINNs. This approach differs from TSB by not relying on traditional solvers for ground-truth data, and from IWR by avoiding direct regression in the weight space. We notice a similar trend in rank analysis, with the best-ranked hypernetwork in POI performing comparably (1.04× worse) to the best in TSB, but significantly outperforming (3-4 orders) INR-regressor hypernetworks. However,

Figure 3: Hypernetwork based LoRA experiments. We have 3 benchmarks, TSB refers to HLoRA trained with analytical ground truths, IWR refers to INR regression on PINNs, POI refers to HLoRA trained on labels generated by PINNs, as elaborated in Sec 4. * refers to the case where we don't perform LoRA weight decomposition and the hypernetwork predicts the entire base network.

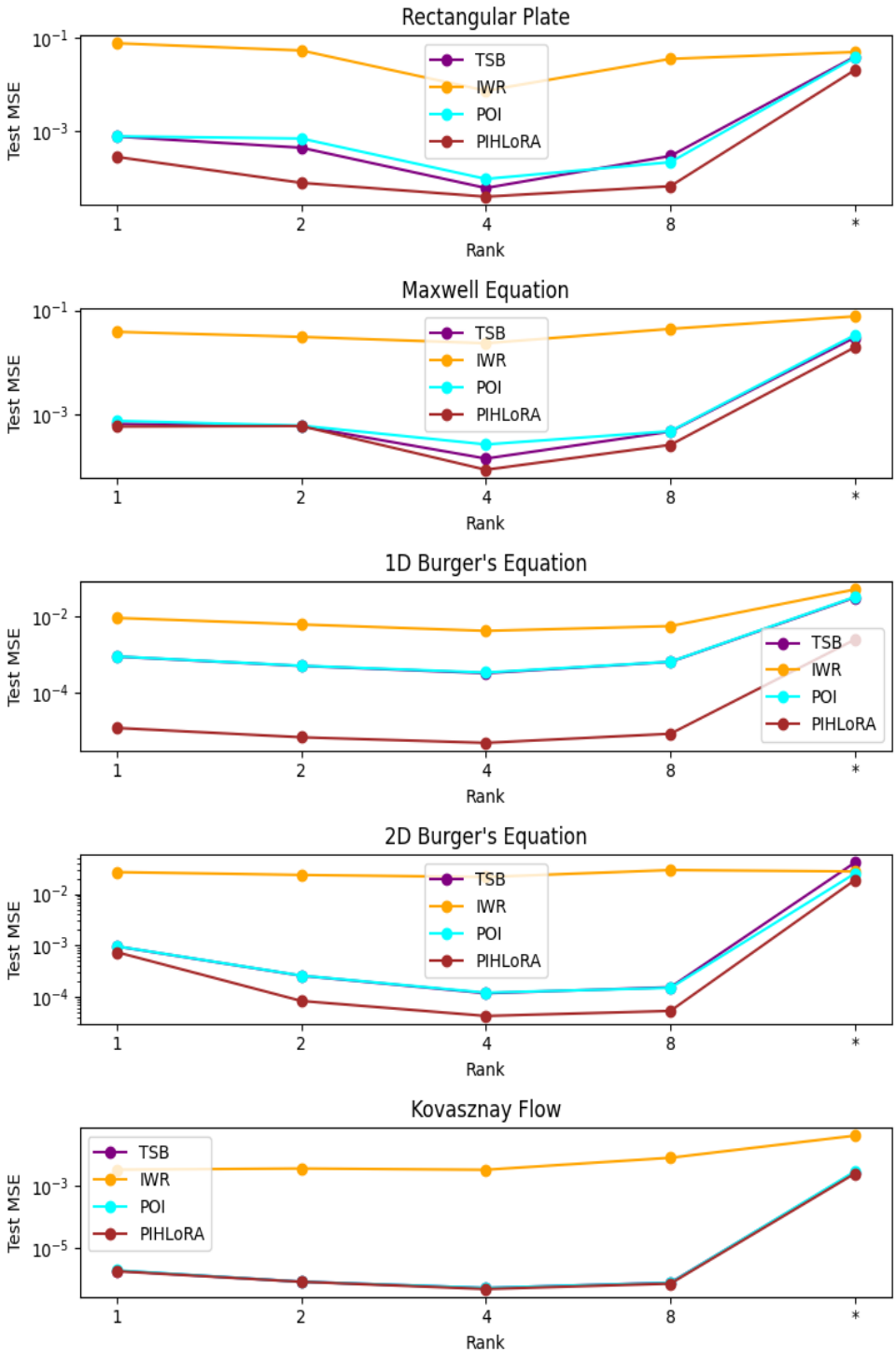

Table 1: Inference-time of the architectures in seconds

Architecture	Rectangular Plate	Maxwell	1D-Burger's	Navier Stokes (Kovasznay flow)	2D Burger's
PINN	38000	18000	8400	6600	9900
F-PINN	31350	12600	7000	4400	8415
L-PINN	3780	2200	600	720	602
PIHLoRA	1.3	1.2	1.1	1.2	1.2

like TSB, POI shows inferior performance (6.51×,3.96×,1.36×, 4.19×, and 463× worse) compared to rank-4 L-PINNs in Kovasznay flow, 2D Burger's, and 1D Burger's, respectively. The challenges here include computational feasibility in training multiple PINNs and poor generalization with physics constraints violations.

To overcome these issues, our final experiment employs Physics-Informed Hypernetworks with Low-Rank Adaptation (PIHLoRA), learning hypernetworks in a physics-informed manner. This approach negates the necessity for pre-trained data or neural surrogate samples, as seen in TSB and POI. We observe that Rank-4 PIHLoRA outperform their counterparts in 1D-Burger's and 2D-Burger's equations by 2-3 orders of magnitude, and only marginally underperform (1.97× worse) than rank-4 L-PINNs in test examples. This indicates that PIHLoRA can generalize effectively across test tasks, offering inference-time advantages and adhering to physics constraints.

We tabulate the inference time for our architectures in Table 1. Across all examples, we observe PIHLoRA to be around 2-3 orders of magnitude faster than vanilla architectures, as the entire computational cost is transferred to one-time train cost, and we simply predict the weights for a test-time, without having to finetune. The best-ranked L-PINNs are $10.25\times$ faster than F-PINNs and $9.16\times$ faster than vanilla-PINNs, due to training lower number of parameters and incorporating pre-trained information from an already trained base-task.

There has also been significant recent interest in using neural network to approximate operators in order to obtain fast solvers of PDEs. Early approaches along this line, such as [50] assumed a finite dimensional fixed resolution mesh for approximating the solution. However recent advances on Neural Operators [51, 52] claim mesh-free learning by utilizing a single set of network parameters for different discretizations. These approaches suffer from two main drawbacks (1) Its not possible to synthesize or record the data needed to facilitate data driven operator learning. Therefore papers like [53–55], have looked into use of physics loss for advancing operator learning approaches to circumstances when we don't have data. (2) Despite the claims of [52], we expect that discretization to cause artifacts, and implicit neural representation will lead to better results. While works like [56] have looked at applying FNOs to irregular geometries, currently there is no literature which models the input and output to an FNO with a neural surrogate.

6 Conclusions

We use Low-ranked adaptation for PINNs to quickly adapt solutions of parameterized PDEs from one instance to another. We investigate the importance of choice of rank of decomposed tensors, and conclude there exists an optimal rank for tensor decomposition, lowering which leads to reduced representation capacity and larger training time for the newer PDE instance. Increasing the rank leads to overfitting and a slight drop in performance at test-time. We note, the optimal low-rank adapted PINN converges faster and outperforms PINNs trained from random initialization or finetuned PINNs. Next, we scale L-PINNs to HyperLoRA-based PINNs to further reduce inference time. Rank analysis of L-PINNs extend to HLoRA as well. HLoRA benchmarks have the drawback of requiring pre-existing ground-truth data which is scarce, and have inferior generalization as the underlying physics isn't captured. Utilizing a physics-loss to train the LoRA-based Hypernetworks (PIHLoRA) leads to improved generalization and comparable performance with instance-wise L-PINNs, while retaining the advantage in inference-time.

References

[1] Prajith Pillai, Anirban Chaudhari, Parama Pal, and Beena Rai. Physics-informed neural network for inversely predicting effective electric permittivities of metamaterials. In *Proceedings of the 35th Neural Information Proceeding Systems, Machine Learning and the Physical Sciences Workshop*, 2021.

[2] Zhiwei Fang and Justin Zhan. Deep physical informed neural networks for metamaterial design. *IEEE Access*, PP:1–1, 12 2019.

[3] S Hamzah, MI Azis, and AK Amir. Numerical solutions to anisotropic bvps for quadratically graded media governed by a helmholtz equation. In *IOP Conference Series: Materials Science and Engineering*, volume 619, page 012060. IOP Publishing, 2019.

[4] N La Nafie, MI Azis, et al. Numerical solutions to bvps governed by the anisotropic modified helmholtz equation for trigonometrically graded media. In *IOP Conference Series: Materials Science and Engineering*, volume 619, page 012058. IOP Publishing, 2019.

[5] P Taba, S Toaha, MI Azis, et al. Numerical solutions to helmholtz equation of anisotropic functionally graded materials. In *Journal of Physics: Conference Series*, volume 1341, page 082012. IOP Publishing, 2019.

[6] Mohammad Vahab, Ehsan Haghighat, Maryam Khaleghi, and Nasser Khalili. A physics informed neural network approach to solution and identification of biharmonic equations of elasticity, 2021.

[7] Taniya Kapoor, Hongrui Wang, Alfredo Núñez, and Rolf Dollevoet. Physics-informed neural networks for solving forward and inverse problems in complex beam systems. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–15, 2023.

[8] Rajat Arora, Pratik Kakkar, Biswadip Dey, and Amit Chakraborty. Physics-informed neural networks for modeling rate- and temperature-dependent plasticity, 2022.

[9] Katayoun Eshkofti and Seyed Mahmoud Hosseini. A gradient-enhanced physics-informed neural network (gpinn) scheme for the coupled non-fickian/non-fourierian diffusion-thermoelasticity analysis: A novel gpinn structure. *Engineering Applications of Artificial Intelligence*, 126:106908, 2023.

[10] Katayoun Eshkofti and Seyed Mahmoud Hosseini. The novel pinn/gpinn-based deep learning schemes for non-fickian coupled diffusion-elastic wave propagation analysis. *Waves in Random and Complex Media*, pages 1–24, 2023.

[11] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

[12] Zewei Sun, Honghan Du, Chunfu Miao, and Qingzhi Hou. A physics-informed neural network based simulation tool for reacting flow with multicomponent reactants. *Advances in Engineering Software*, 185:103525, 2023.

[13] Tianfeng Zhou, Jiwang Yan, Jun Masuda, and Tsunemoto Kuriyagawa. Investigation on the viscoelasticity of optical glass in ultraprecision lens molding process. *Journal of materials processing technology*, 209(9):4484–4489, 2009.

[14] Ernesto Medina, Terence Hwa, Mehran Kardar, and Yi-Cheng Zhang. Burgers equation with correlated noise: Renormalization-group analysis and applications to directed polymers and interface growth. *Physical Review A*, 39(6):3053, 1989.

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

[16] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one, 2022.

[17] Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. Implicit neural spatial representations for time-dependent pdes, 2023.

[18] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

[19] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[20] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[21] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Wei Wei, Tingbo Hou, Yael Pritch, Neal Wadhwa, Michael Rubinstein, and Kfir Aberman. Hyperdreambooth: Hypernetworks for fast personalization of text-to-image models. *arXiv preprint arXiv:2307.06949*, 2023.

[22] Hamish Ivison, Akshita Bhagia, Yizhong Wang, Hannaneh Hajishirzi, and Matthew Peters. Hint: Hypernetwork instruction tuning for efficient zero- few-shot generalisation, 2023.

[23] Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals, 2023.

[24] Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12799–12807, 2023.

[25] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

[26] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

[27] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[28] Yu Yu, Chao-Han Huck Yang, Jari Kolehmainen, Prashanth G Shivakumar, Yile Gu, Sungho Ryu, Roger Ren, Qi Luo, Aditya Gourav, I-Fan Chen, et al. Low-rank adaptation of large language model rescoring for parameter-efficient speech recognition. *arXiv preprint arXiv:2309.15223*, 2023.

[29] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

[30] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhensu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

[31] Adam X Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023.

[32] Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. *arXiv preprint arXiv:2310.17513*, 2023.

[33] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

[34] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*, 2023.

[35] Nafise Sadat Moosavi, Quentin Delfosse, Kristian Kersting, and Iryna Gurevych. Adaptable adapters. *arXiv preprint arXiv:2205.01549*, 2022.

[36] Zih-Ching Chen, Yu-Shun Sung, and Hung-yi Lee. Chapter: Exploiting convolutional neural network adapters for self-supervised speech models. In *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, pages 1–5. IEEE, 2023.

[37] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*, 2020.

[38] Shengrui Li, Xueting Han, and Jing Bai. Adaptergnn: Efficient delta tuning improves generalization ability in graph neural networks. *arXiv preprint arXiv:2304.09595*, 2023.

[39] Apostolos F Psaros, Kenji Kawaguchi, and George Em Karniadakis. Meta-learning pinn loss functions. *Journal of Computational Physics*, 458:111121, June 2022.

[40] Xu Liu, Xiaoya Zhang, Wei Peng, Weien Zhou, and Wen Yao. A novel meta-learning initialization method for physics-informed neural networks. *Neural Computing and Applications*, 34(17):14511–14534, May 2022.

[41] Michael Penwarden, Shandian Zhe, Akil Narayan, and Robert M. Kirby. Physics-informed neural networks (pinns) for parameterized pdes: A metalearning approach, 2021.

[42] Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *arXiv preprint arXiv:2306.06955*, 2023.

[43] Fazilet Gokbudak, Alejandro Sztrajman, Chenliang Zhou, Fangcheng Zhong, Rafal Mantiuk, and Cengiz Oztireli. Hypernetworks for generalizable brdf estimation, 2023.

[44] Filipe de Avila Belbute-Peres, Yi fan Chen, and Fei Sha. Hyperpinn: Learning parameterized differential equations with physics-informed hypernetworks, 2021.

[45] Ritam Majumdar, Vishal Jadhav, Anirudh Deodhar, Shirish Karande, Lovekesh Vig, and Venkataramana Runkana. Physics informed symbolic networks, 2022.

[46] Amirali Molaei, Amirhossein Aminimehr, Armin Tavakoli, Amirhossein Kazerouni, Bobby Azad, Reza Azad, and Dorit Merhof. Implicit neural representation in medical imaging: A comparative survey, 2023.

[47] Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, GA Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Carlberg, and Eitan Grinspun. Crom: Continuous reduced-order modeling of pdes using implicit neural representations. *arXiv preprint arXiv:2206.02607*, 2022.

[48] W Zhang, W Diab, and M Al Kobaisi. Physics informed neural networks for solving highly anisotropic diffusion equations. In *ECMOR 2022*, volume 2022, pages 1–15. European Association of Geoscientists & Engineers, 2022.

[49] Ritam Majumdar, Vishal Jadhav, Anirudh Deodhar, Shirish Karande, Lovekesh Vig, and Venkataramana Runkana. Real-time health monitoring of heat exchangers using hypernetworks and pinns, 2022.

[50] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 481–490, New York, NY, USA, 2016. Association for Computing Machinery.

[51] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

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

[53] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations, 2023.

[54] Ritam Majumdar, Amey Varhade, Shirish Karande, and Lovekesh Vig. Can physics informed neural operators self improve?, 2023.

[55] Navaneeth N, Tapas Tripura, and Souvik Chakraborty. Physics informed wno, 2023.

[56] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries, 2022.

A Appendix

The appendix is organized as follows. Section A.1 describes the PDE information, A.2 refers to the training and hyperparameter details of our experiments. A.3 tabulates the rank-analysis results, followed by Hypernetwork experiment result tables in A.4.

A.1 PDE Information

A.1.1 Free Vibration of Rectangular Plate

The fourth order governing PDE system describing the out of plane motion of a plate is described as:

$$D(\frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 y^2} + \frac{\partial^4 u}{\partial y^4}) = -\rho\frac{\partial^2 u}{\partial t^2} \tag{8}$$

Here $D = Eh^3/12(1-\nu^2)$ is the flexural stiffness, E is Young's modulus, h as thickness of the plate and ν is the Poisson's ratio. The plate is subjected to the following initial and boundary conditions:

$$u = 0, \;\; \frac{\partial^2 u}{\partial n^2} = 0 \;\text{ for }\; x \in \Gamma_u, \qquad u = sin(\pi x)sin(\pi y), \;\; u_t = 0 \;\text{ for }\; t = 0 \tag{9}$$

Here $\Omega \in [0,1] \times [0,1]$,$t \in [0, 0.1]$. We consider $D \in [300, 500]$ and $\rho \in [2000, 4000] kg/m^3$ for HyperLoRA experiments, and $D = 398.222$ and $\rho = 2700 kg/m^3$ for LoRA experiments.

A.1.2 Two dimensional Maxwell's equation

We consider a two-dimensional Maxwell equation in heterogeneous medium whose governing equations are given by:

$$\epsilon\frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y} \qquad \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -\mu\frac{\partial H_z}{\partial t} \qquad \epsilon\frac{\partial E_y}{\partial t} = -\frac{\partial H_z}{\partial x} \tag{10}$$

Here, $\Omega_1 \in [0, \pi] \times [0, 2\pi]$, $\Omega_2 \in [\pi, 2\pi] \times [0, 2\pi]$, $t \in [0, 2]$ with $x = \pi$ is the interface. ϵ and μ refers to Electric permeability and magnetic permeability respectively. E and H are Electric and Magnetic field vectors respectively. The analytical solutions of the given PDE system is given by:

$$E_x = \frac{k_y cos(\omega t)cos(k_x x)sin(k_y y)}{\epsilon\sqrt{\mu}\omega} \; E_y = \frac{k_x cos(\omega t)sin(k_x x)cos(k_y y)}{-\epsilon\sqrt{\mu}\omega} \; H_z = \frac{sin(\omega t)cos(k_x x)cos(k_y y)}{\sqrt{\mu}} \tag{11}$$

In our LoRA experiments, we consider $\mu = 1, k_y = 2, \epsilon(x) = 2$ if $x \in \Omega_1, \epsilon(x) = 5$ if $x \in \Omega_2, k_x(x) = 2$ if $x \in \Omega_1, k_x(x) = 4$ if $x \in \Omega_2$. In our HyperLoRA experiments, $k_y \in [1.5, 2.5], \epsilon(x) = [1, 3]$ if $x \in \Omega_1, \epsilon(x) = [4, 6]$ if $x \in \Omega_2, k_x(x) = [1, 3]$ if $x \in \Omega_1, k_x(x) = [3, 5]$ if $x \in \Omega_2$. k_x, k_y represent wave numbers, $\omega = \frac{k_x^2 + k_y^2}{\mu\epsilon}$ represent the angular frequency of the wave.

A.1.3 One-dimensional Burger's equation

The governing PDE system of one-dimensional Burger's equation is given by:

$$u_t + u_x^2(x,t)/2 = \nu u_{xx} \qquad u(x,0) = u_0(x) \tag{12}$$

Here, $\Omega \in [0,1]$,$\nu = 0.01$ refers to the viscosity of the system, and $u_0 \in L^2_{per}((0,1); R)$ refers to the initial condition sampled from μ where $\mu = N(0, 625(-\Delta + 25I)^{-2})$, sampled using a Gaussian Random Field.

A.1.4 Two-dimensional coupled viscous Burger's equation

The governing equations of two-dimensional coupled viscous Burger's equation is given by:

$$u_t + u * u_x + v * u_y = \nu * (u_{xx} + u_{yy}) \qquad v_t + u * v_x + v * v_y = \nu * (v_{xx} + v_{yy}) \tag{13}$$

The initial and boundary conditions are sampled from the true analytical solutions of the PDE, given by:

$$u(x,y,t) = \frac{3}{4} + \frac{1}{4(1+exp(\frac{(-4x+4y-t)}{32\nu}))} \qquad v(x,y,t) = \frac{3}{4} - \frac{1}{4(1+exp(\frac{(-4x+4y-t)}{32\nu}))} \quad (14)$$

Here $\Omega \in [0,1]^2$, $t \in [0,1]$, ν refers to the viscosity of the flow. We consider $\nu \in [1e^{-4}, 1e^{-3}]$ for HyperLoRA experiments, and use $\nu = 5e^{-4}$ for LoRA experiments.

A.1.5 Navier-Stokes: Kovasznay flow

Navier-Stokes: Kovasznay flow is a two-dimensional steady state laminar flow. The governing Partial Differential equations are given by:

$$u_x + v_y = 0 \quad u*u_x + v*u_y = -p_x + (u_{xx} + u_{yy})/Re \quad u*v_x + v*v_y = -p_y + (v_{xx} + v_{yy})/Re \quad (15)$$

The boundary conditions for the equation are sampled from the true analytical solutions of the PDE, given by:

$$u_{true} = 1 - e^{\lambda x} cos(2\pi y) \qquad v_{true} = \lambda e^{\lambda x} sin(2\pi y)/2\pi \qquad p_{true} = (1 - e^{2\lambda x})/2 \quad (16)$$

Here, $\Omega \in [0,1]^2$, $\lambda = \frac{Re}{2} - \sqrt{\frac{Re^2}{4} + 4\pi^2}$, where Re refers to the Reynold's number of the system. We consider $Re \in [20, 100]$ for HyperLoRA experiments and $Re = 40$ for LoRA experiments.

A.2 Training and Hyperparameter Details

A.2.1 Rank-Analysis Experiments

In our rank-analysis experiments, we employ a Multi-layer Perceptron (MLP) with an architecture of [inputs-64*4-outputs] and use hyperbolic tangent activation functions, specifically tailored for training PINNs. The training of the base PINN is conducted using the Adam optimizer over 40k epochs, with an initial learning rate of 1e-3 for the first 10k epochs. This is followed by a multiplicative decay factor of 0.1, descending to a minimum learning rate of $1e^{-7}$. The training process is terminated if there is no improvement in validation loss for 1k consecutive epochs. Further refinement of the trained PINNs is done using an L-BFGS optimizer. This training protocol, including the optimizer and learning rate schedule, is applied to F-PINNs and L-PINNs. In the Rectangular Plate Vibration example, we discretize the domain into a $20 \times 20 \times 10$ grid across both spatial and temporal dimensions. For Maxwell's equations, our setup includes 400 boundary condition points (100 per boundary), 1k initial condition points, and 15k uniformly distributed collocation points. For the 1D-Burger's equation, we use 128 initial condition points and 10k collocation points for training. In the case of the 2D-Burger's equation, the setup consists of 400 boundary condition points (100 on each side of the square grid), 500 initial condition points, and 10k collocation points. Lastly, for Navier-Stokes: Kovasznay flow, we employ a 101×101 grid for the spatial domain and sample 2,601 collocation points, alongside 320 boundary condition points (80 for each face of the grid).

A.2.2 Hypernetwork-Based Experiments

In our hypernetwork-based experiments, we use an MLP with an architecture of [inputs- 512*2 - 256*2 - 128*2 - outputs], utilizing hyperbolic tangent activation. The dimensionality of the inputs varies depending on the task: 3 for Maxwell's equation, 2 for the rectangular plate, 1 for Navier-Stokes (Reynold's number Re), 1 for 2D-Burger's (viscosity ν), and 128 for 1D-Burger's (initial condition $u_0(x)$ discretized at 128 positions). The outputs pertain to the number of parameters being predicted, with specific details provided in Tables 5, 7, and 6. All hypernetworks are trained using the Adam optimizer for 15k epochs, starting with a learning rate of $1e^{-3}$ for the initial 5k epochs and implementing a decay of 0.1 every 3k epochs. Across all examples, we employ 20 validation and test tasks. The number of training tasks is set to 20 for the Rectangular Plate, 40 for Maxwell's Equations, 100 for 1D-Burger's, 20 for 2D-Burger's, and 20 for the Navier-Stokes equations. All experiments were conducted on an Nvidia P100 GPU, featuring 16 GB of GPU memory and a 1.32 GHz memory clock, using the Pytorch framework.

A.3 L-PINN (Rank analysis) results for our experiments

Rectangular Plate	Rank	#	Error	Time	Epochs
PINNs		21187	$6.64e^{-5}$	0.95	40k
F-PINNs		21187	$4.24e^{-5}$	0.95	33k
L-PINNs	1	645	$5.18e^{-5}$	0.06	40k
	2	1290	$4.01e^{-5}$	0.13	33k
	4	**2580**	$\mathbf{1.42}e^{-5}$	**0.21**	**18k**
	8	5160	$2.28e^{-5}$	0.34	21k
	16	10320	$3.99e^{-5}$	0.43	25k
	32	20640	$5.01e^{-5}$	0.61	27k
Maxwell	Rank	#	Error	Time	Epochs
PINNs		21188	$3.65e^{-4}$	0.36	50k
F-PINNs		21188	$2.77e^{-4}$	0.36	35k
L-PINNs	1	646	$3.28e^{-4}$	0.03	50k
	2	1292	$1.45e^{-4}$	0.07	35k
	4	**2584**	$\mathbf{6.61}e^{-5}$	**0.10**	**22k**
	8	5168	$9.05e^{-5}$	0.14	26k
	16	10336	$1.84e^{-4}$	0.17	32k
	32	20672	$2.02e^{-4}$	0.20	32k
1D-Burger's	Rank	#	Error	Time	Epochs
PINNs		21057	$4.54e^{-6}$	0.28	30k
F-PINNs		21057	$1.93e^{-6}$	0.28	25k
L-PINNs	1	643	$1.95e^{-6}$	0.01	30k
	2	1286	$1.54e^{-6}$	0.03	25k
	4	**2572**	$\mathbf{7.00}e^{-7}$	**0.05**	**12k**
	8	5144	$8.89e^{-7}$	0.09	12.5k
	16	10288	$1.00e^{-6}$	0.12	20k
	32	20576	$1.03e^{-6}$	0.16	20k
2D-Burger's	Rank	#	Error	Time	Epochs
PINNs		21187	$3.25e^{-5}$	0.33	30k
F-PINNs		21187	$2.69e^{-5}$	0.33	25.5k
L-PINNs	1	645	$3.81e^{-4}$	0.02	30k
	2	1290	$7.02e^{-5}$	0.04	25k
	4	**2580**	$\mathbf{1.17}e^{-5}$	**0.05**	**12k**
	8	5160	$2.73e^{-5}$	0.10	13k
	16	10320	$2.89e^{-5}$	0.14	20k
	32	20640	$2.94e^{-5}$	0.17	20k
Kovasznay	Rank	#	Error	Time	Epochs
PINNs		21187	$1.05e^{-6}$	0.22	30k
F-PINNs		21187	$6.36e^{-7}$	0.22	20k
L-PINNs	1	645	$8.16e^{-7}$	0.02	30k
	2	1290	$6.68e^{-7}$	0.04	20k
	4	**2580**	$\mathbf{3.85}e^{-7}$	**0.06**	**12k**
	8	5160	$5.59e^{-7}$	0.09	12.5k
	16	10320	$9.35e^{-7}$	0.14	20k
	32	20640	$9.74e^{-7}$	0.16	30k

Table 2: Low-ranked Adaptation results. Here, F-PINNs refer to PINNs finetuned directly, L-PINNs refer to Low-ranked adaptation based PINNs. Error refers to MSE averaged over 20 test-tasks, to be used in Tables 5,7,6. Time refers to training time per epoch in seconds. # refers to number of parameters updated in training-time.

A.4 HLoRA results for our experiments

Table 3: Mean squared error of Hypernetwork-based LoRA experiments for Rectangular Plate

	Rank	# Param	Train	Valid	Test	Time (per epoch)
TSB	1	645	$6.82e^{-4}$	$6.97e^{-4}$	$7.56e^{-4}$	0.24
	2	1290	$4.49e^{-4}$	$4.22e^{-4}$	$4.34e^{-4}$	0.35
	4	**2580**	$\mathbf{6.03}e^{-5}$	$\mathbf{5.62}e^{-5}$	$\mathbf{5.89}e^{-5}$	**0.59**
	8	5160	$2.67e^{-4}$	$2.53e^{-4}$	$2.88e^{-4}$	0.88
	*	20640	$1.34e^{-2}$	$2.15e^{-2}$	$3.97e^{-2}$	1.16
IWR	1	645	$8.12e^{-2}$	$8.56e^{-2}$	$7.71e^{-2}$	0.031
	2	1290	$5.49e^{-2}$	$5.55e^{-2}$	$5.39e^{-2}$	0.044
	4	**2580**	$\mathbf{8.01}e^{-3}$	$\mathbf{7.85}e^{-3}$	$\mathbf{7.34}e^{-3}$	**0.059**
	8	5160	$3.92e^{-2}$	$3.45e^{-2}$	$3.56e^{-2}$	0.082
	*	20640	$1.06e^{-2}$	$3.32e^{-2}$	$5.00e^{-2}$	0.29
POI	1	645	$8.15e^{-4}$	$7.26e^{-4}$	$7.64e^{-4}$	0.24
	2	1290	$6.74e^{-4}$	$7.17e^{-4}$	$6.85e^{-4}$	0.35
	4	**2580**	$\mathbf{8.65}e^{-5}$	$\mathbf{8.48}e^{-5}$	$\mathbf{9.23}e^{-5}$	**0.60**
	8	5160	$1.97e^{-4}$	$2.88e^{-4}$	$2.12e^{-4}$	0.88
	*	20640	$1.34e^{-2}$	$2.14e^{-2}$	$3.78e^{-2}$	1.15
PIHLoRA	1	645	$3.67e^{-4}$	$3.93e^{-4}$	$2.75e^{-4}$	0.84
	2	1290	$5.46e^{-5}$	$5.98e^{-5}$	$7.60e^{-5}$	2.04
	4	**2580**	$\mathbf{3.15}e^{-5}$	$\mathbf{4.84}e^{-5}$	$\mathbf{3.83}e^{-5}$	**4.12**
	8	5160	$4.54e^{-5}$	$5.58e^{-5}$	$6.44e^{-5}$	5.37
	*	20640	$4.52e^{-2}$	$1.91e^{-2}$	$2.05e^{-2}$	14.26

Table 4: Mean squared error of Hypernetwork-based LoRA experiments for Maxwell equation

	Rank	# Param	Train	Valid	Test	Time (per epoch)
TSB	1	646	$6.65e^{-4}$	$6.34e^{-4}$	$6.52e^{-4}$	0.16
	2	1292	$5.43e^{-4}$	$5.70e^{-4}$	$5.91e^{-4}$	0.26
	4	**2584**	$\mathbf{1.19}e^{-4}$	$\mathbf{1.35}e^{-4}$	$\mathbf{1.40}e^{-4}$	**0.40**
	8	5168	$3.77e^{-4}$	$4.14e^{-4}$	$4.64e^{-4}$	0.71
	*	21272	$1.75e^{-2}$	$2.29e^{-2}$	$3.01e^{-2}$	3.04
IWR	1	646	$3.35e^{-2}$	$4.16e^{-2}$	$3.86e^{-2}$	0.04
	2	1292	$3.54e^{-2}$	$3.75e^{-2}$	$3.09e^{-2}$	0.08
	4	**2584**	$\mathbf{2.12}e^{-2}$	$\mathbf{2.09}e^{-2}$	$\mathbf{2.33}e^{-2}$	**0.15**
	8	5168	$4.04e^{-2}$	$4.17e^{-2}$	$4.41e^{-2}$	0.23
	*	21272	$8.19e^{-2}$	$7.65e^{-2}$	$7.68e^{-2}$	1.06
POI	1	646	$7.09e^{-4}$	$6.89e^{-4}$	$7.45e^{-4}$	0.16
	2	1292	$5.96e^{-4}$	$5.81e^{-4}$	$6.04e^{-4}$	0.26
	4	**2584**	$\mathbf{2.05}e^{-4}$	$\mathbf{2.32}e^{-4}$	$\mathbf{2.62}e^{-4}$	**0.40**
	8	5168	$4.16e^{-4}$	$3.99e^{-4}$	$4.71e^{-4}$	0.71
	*	21272	$3.05e^{-2}$	$3.51e^{-2}$	$3.36e^{-2}$	3.05
PIHLoRA	1	646	$5.16e^{-4}$	$5.55e^{-4}$	$5.79e^{-4}$	0.57
	2	1292	$4.74e^{-4}$	$5.03e^{-4}$	$5.92e^{-4}$	1.65
	4	**2584**	$\mathbf{9.05}e^{-5}$	$\mathbf{8.29}e^{-5}$	$\mathbf{8.52}e^{-5}$	**3.88**
	8	5168	$2.32e^{-4}$	$2.96e^{-4}$	$2.57e^{-4}$	4.78
	*	21272	$3.76e^{-2}$	$4.17e^{-2}$	$1.95e^{-2}$	12.53

Table 5: Mean squared error of Hypernetwork-based LoRA experiments for 1D Burger's equation.

	Rank	# Param	Train	Valid	Test	Time (per epoch)
TSB	1	643	$7.25e^{-4}$	$1.09e^{-3}$	$8.78e^{-4}$	1.19
	2	1286	$2.69e^{-4}$	$4.67e^{-4}$	$4.98e^{-4}$	1.56
	4	**2572**	$\mathbf{3.02}e^{-4}$	$\mathbf{3.47}e^{-4}$	$\mathbf{3.22}e^{-4}$	**1.84**
	8	5144	$5.69e^{-4}$	$6.11e^{-4}$	$6.37e^{-4}$	2.25
	*	21057	$1.96e^{-2}$	$3.70e^{-2}$	$3.14e^{-2}$	17.26
IWR	1	643	$8.82e^{-3}$	$8.54e^{-3}$	$9.13e^{-3}$	0.81
	2	1286	$6.19e^{-3}$	$6.15e^{-3}$	$6.15e^{-3}$	1.05
	4	**2572**	$\mathbf{4.24}e^{-3}$	$\mathbf{4.43}e^{-3}$	$\mathbf{4.17}e^{-3}$	**1.36**
	8	5144	$5.43e^{-3}$	$5.94e^{-3}$	$5.56e^{-3}$	1.74
	*	21057	$3.64e^{-2}$	$4.90e^{-2}$	$5.12e^{-2}$	10.14
POI	1	643	$7.18e^{-4}$	$1.12e^{-3}$	$8.89e^{-4}$	1.19
	2	1286	$2.86e^{-4}$	$4.75e^{-4}$	$5.05e^{-4}$	1.56
	4	**2572**	$\mathbf{3.12}e^{-4}$	$\mathbf{3.55}e^{-4}$	$\mathbf{3.35}e^{-4}$	**1.84**
	8	5144	$5.58e^{-4}$	$6.16e^{-4}$	$6.44e^{-4}$	2.25
	*	21057	$2.02e^{-2}$	$3.84e^{-2}$	$3.26e^{-2}$	17.26
PIHLoRA	1	643	$9.64e^{-6}$	$1.45e^{-5}$	$1.17e^{-5}$	3.15
	2	1286	$4.02e^{-6}$	$6.22e^{-6}$	$6.64e^{-6}$	4.35
	4	**2572**	$\mathbf{3.67}e^{-6}$	$\mathbf{4.82}e^{-6}$	$\mathbf{4.74}e^{-6}$	**5.75**
	8	5144	$7.19e^{-6}$	$8.19e^{-6}$	$8.23e^{-6}$	8.45
	*	21057	$2.58e^{-3}$	$4.29e^{-3}$	$2.47e^{-3}$	67.6

Table 6: Mean squared error of Hypernetwork-based LoRA experiments for Kovasznay flow

	Rank	# Param	Train	Valid	Test	Time (per epoch)
TSB	1	645	$2.19e^{-6}$	$1.39e^{-6}$	$1.87e^{-6}$	0.16
	2	1290	$4.45e^{-7}$	$9.16e^{-7}$	$8.31e^{-7}$	0.21
	4	**2580**	$\mathbf{4.08}e^{-7}$	$\mathbf{5.16}e^{-7}$	$\mathbf{5.23}e^{-7}$	**0.29**
	8	5160	$6.46e^{-7}$	$6.99e^{-7}$	$7.75e^{-7}$	0.37
	*	21187	$4.38e^{-4}$	$3.60e^{-3}$	$2.66e^{-3}$	2.15
IWR	1	645	$4.17e^{-3}$	$3.29e^{-3}$	$3.22e^{-3}$	0.05
	2	1290	$3.95e^{-3}$	$3.59e^{-3}$	$3.49e^{-3}$	0.07
	4	**2580**	$\mathbf{2.95}e^{-3}$	$\mathbf{3.06}e^{-3}$	$\mathbf{3.19}e^{-3}$	**0.101**
	8	5160	$4.04e^{-3}$	$5.19e^{-3}$	$7.72e^{-3}$	0.12
	*	21187	$1.17e^{-2}$	$2.18e^{-2}$	$1.95e^{-2}$	0.73
POI	1	645	$2.19e^{-6}$	$1.39e^{-6}$	$1.91e^{-6}$	0.16
	2	1290	$4.28e^{-7}$	$9.03e^{-7}$	$8.29e^{-7}$	0.21
	4	**2580**	$\mathbf{4.11}e^{-7}$	$\mathbf{5.04}e^{-7}$	$\mathbf{5.26}e^{-7}$	**0.29**
	8	5160	$6.28e^{-7}$	$7.15e^{-7}$	$7.64e^{-7}$	0.37
	*	21187	$4.47e^{-4}$	$3.66e^{-3}$	$2.82e^{-3}$	2.15
PIHLoRA	1	645	$2.22e^{-6}$	$1.35e^{-6}$	$1.80e^{-6}$	0.63
	2	1290	$4.36e^{-7}$	$9.15e^{-7}$	$8.25e^{-7}$	0.84
	4	**2580**	$\mathbf{4.15}e^{-7}$	$\mathbf{5.52}e^{-7}$	$\mathbf{4.84}e^{-7}$	**1.15**
	8	5160	$5.93e^{-7}$	$6.64e^{-7}$	$7.22e^{-7}$	1.45
	*	21187	$3.17e^{-4}$	$2.91e^{-3}$	$4.42e^{-3}$	9.43

Table 7: Mean squared error of Hypernetwork-based LoRA experiments for 2D Burger's equation

	Rank	# Param	Train	Valid	Test	Time (per epoch)
TSB	1	644	$3.05e^{-4}$	$4.44e^{-4}$	$9.52e^{-4}$	0.093
	2	1288	$2.97e^{-4}$	$3.26e^{-4}$	$2.55e^{-4}$	0.131
	4	**2576**	$\mathbf{1.46}e^{-4}$	$\mathbf{1.50}e^{-4}$	$\mathbf{1.16}e^{-4}$	**0.171**
	8	5152	$1.81e^{-4}$	$1.66e^{-4}$	$1.52e^{-4}$	0.241
	*	21122	$4.45e^{-2}$	$3.99e^{-2}$	$4.12e^{-2}$	1.91
IWR	1	644	$3.17e^{-2}$	$3.22e^{-2}$	$2.68e^{-2}$	0.031
	2	1288	$2.12e^{-2}$	$2.15e^{-2}$	$2.36e^{-2}$	0.044
	4	**2576**	$\mathbf{1.40}e^{-2}$	$\mathbf{1.57}e^{-2}$	$\mathbf{2.15}e^{-2}$	**0.059**
	8	5152	$2.66e^{-2}$	$2.36e^{-2}$	$2.94e^{-2}$	0.082
	*	21122	$3.25e^{-2}$	$3.61e^{-2}$	$2.76e^{-2}$	2.95
POI	1	644	$3.12e^{-4}$	$4.63e^{-4}$	$9.58e^{-4}$	0.093
	2	1288	$2.88e^{-4}$	$3.32e^{-4}$	$2.57e^{-4}$	0.131
	4	**2576**	$\mathbf{1.49}e^{-4}$	$\mathbf{1.61}e^{-4}$	$\mathbf{1.19}e^{-4}$	**0.171**
	8	5152	$1.85e^{-4}$	$1.58e^{-4}$	$1.48e^{-4}$	0.241
	*	21122	$1.82e^{-2}$	$2.06e^{-2}$	$2.52e^{-2}$	1.91
PIHLoRA	1	644	$5.52e^{-4}$	$8.26e^{-4}$	$7.35e^{-4}$	0.375
	2	1288	$6.14e^{-5}$	$7.66e^{-5}$	$8.22e^{-5}$	0.525
	4	**2576**	$\mathbf{2.89}e^{-5}$	$\mathbf{4.26}e^{-5}$	$\mathbf{4.19}e^{-5}$	**0.69**
	8	5152	$5.39e^{-5}$	$5.87e^{-5}$	$5.25e^{-5}$	0.975
	*	21122	$6.34e^{-3}$	$2.76e^{-2}$	$1.88e^{-2}$	7.85